UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Davor Pobega

# Primer izboljšave odzivnosti trinivojske aplikacije

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2016

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Davor Pobega

# Primer izboljšave odzivnosti trinivojske aplikacije

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: izr. prof. dr. Danijel Skočaj

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Odzivnost je z uporabniškega vidika zelo pomembna za učinkovito uporabo aplikacije. Zaradi različnih vzrokov se lahko odzivnost aplikacije z leti uporabe spreminja, zato je zelo pomembno, da odzivnost spremljamo ter po potrebi ustrezno ukrepamo. V diplomski nalogi opišite primer izboljšave odzivnosti trinivojske aplikacije. Osredotočite se na izbrano aplikacijo ter na možnosti izboljšave odzivnosti z analizo trenutne odzivnosti operacij na predstavitvenem, aplikacijskem in podatkovnem nivoju aplikacije. Predstavite predloge za pohitritev, opišite izvedbo ter prikažite rezultate. Poleg tega identificirajte vzroke za pojavljanje dogodkov čezmerno nizke odzivnosti in jih poskušajte preprečiti ter vzpostavite dolgoročen sistem nadzora odzivnosti aplikacije.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani Davor Pobega sem avtor diplomskega dela z naslovom:

Primer izboljšave odzivnosti trinivojske aplikacije (angl. *A case study of three layer application performance optimization*).

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Danijela Skočaja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu prek univerzitetnega spletnega arhiva.

V Ljubljani, 16. avgusta 2016 Podpis avtorja:

# Kazalo

# Seznam uporabljenih kratic

| Kratica | Angleško | Slovensko |
|---|---|---|
| **SQL** | structured query language | strukturiran povpraševalni jezik za delo s podatkovnimi bazami |
| **SSD** | solid state drive | disk z bliskovnim pomnilnikom |
| **WCF** | windows communication foundation | |
| **SOA** | service oriented architecture | storitveno orientirana arhitektura |

# Povzetek

**Naslov:** Primer izboljšave odzivnosti trinivojske aplikacije

Odzivnost je eden od aspektov kvalitete storitve programske opreme. Če ni nadzorovana, potem se lahko poslabša do točke, kjer postane nezadovoljstvo uporabnikov kritično. To zahteva nepredvidene, obsežne in zahtevne posege v aplikacijo. Namen tega diplomskega dela je opisati tak primer izboljševanja odzivnosti trinivojske aplikacije. Predstavljena je programska arhitektura, uporabljene razvojne tehnologije ter okolje delovanja aplikacije. Pojasnjen je pristop naslavljanja nezadovoljstva uporabnikov. Opisane so aktivnosti za pohitritev izvrševanja kritičnih operacij, vzpostavitve procesa dolgoročnega spremljanja odzivnosti in zmanjševanja verjetnosti pojava dogodkov čezmerno nizke odzivnosti.

**Ključne besede:** trinivojska aplikacija, performančna optimizacija, odzivnost

# Abstract

**Title:** A case study of three layer application responsiveness optimization

Application responsiveness is a quality of service consideration. If neglected for a longer period, it can deteriorate to a point where user's dissatisfaction becomes critical. This may require unplanned, substantial and demanding efforts for responsiveness optimization. The purpose of this thesis is to describe such a case of a three layer application responsiveness optimization. Initially the thesis presents the application's architecture, used developmental technologies and application's environment. It explains the approach that was used to address end users dissatisfaction, details the activities for responsiveness optimization and the importance of responsiveness monitoring. Furthermore it describes the activities for prevention of occasional events of extremely bad application responsiveness.

**Keywords:** three layer application, performance optimization, responsiveness

# Poglavje 1 Uvod

Odzivnost je eden od aspektov kvalitete storitve programske opreme. Ostali aspekti so nadgradljivost, razpoložljivost, obvladljivost, celovitost in varnost [3]. Ti morajo biti uravnovešeni z odzivnostjo in to praviloma pomeni sklepanje arhitekturnih in oblikovnih kompromisov. Vrsta principov snovanja lahko glede na dane cilje odzivnosti in drugih aspektov pripomore k ustrezni arhitekturi in načrtu aplikacije. Pravi čas za upoštevanje teh principov je v fazi snovanja arhitekture in načrta aplikacije. Kasneje v življenjskem ciklu aplikacije so možnosti za spreminjanje in uvedbo teh principov omejene.

Tudi če sta bili arhitektura in načrt aplikacije primerno zastavljeni, se lahko potrebe (npr. zahteve uporabnikov) v nadaljnjih letih življenjskega cikla aplikacije bolj ali manj intenzivno spreminjajo (povečujejo). Pri tem lahko bistveno presežejo nivo začetnih potreb. Razvoj aplikacije, ki sledi tem povečanim potrebam, lahko negativno vpliva na odzivnost posameznih delov ali celotne aplikacije. Še posebej to velja, če aspektu odzivnosti ni ves čas posvečena ustrezna pozornost. V nalogi obravnavamo primer takega razvoja aplikacije. Opisujemo nastalo zmanjšano odzivnost, nezadovoljstvo uporabnikov in motivacijo za reševanje nastale problematike. Ta je razdeljena na tri probleme, in sicer postopni upad odzivnosti (pretežno posledica prej omenjenega načina razvoja), potreba po dolgoročnem spremljanju odzivnosti in obravnava dogodkov čezmerno nizke odzivnosti. Za omenjene tri probleme smo naredili analizo ter podali rešitve in jih ocenili.

## 1.1 O Aplikaciji

Razvoj aplikacije, ki jo obravnavamo v tem delu (v nadaljevanju Aplikacija), se je začel leta 2008. Prva različica je bila nameščena po letu razvoja. V nadaljevanju podajamo opis izbrane arhitekturne osnove in uporabljenih tehnologij.

### *1.1.1 Trinivojska arhitektura*

Aplikacija je implementirana na osnovi trinivojske arhitekture. Model trinivojske arhitekture segmentira logične komponente aplikacije v tri nivoje. Ti nivoji ne odražajo nujno fizičnih lokacij na različnih računalnikih, povezanih z mrežo. Porazdelitev logičnih komponent oziroma

nivojev ne odraža nujno fizične porazdeljenosti (fizične topologije). Obe sta lahko predmet neodvisnih zahtev in specifik.

Logični nivoji so:

- **Predstavitveni nivo**. Namenjen je interakciji med uporabniki in sistemom. Je tudi vstopna/povezovalna točka med sistemom in perifernimi napravami (tiskalnik, ostale naprave). Omogočati mora hiter in učinkovit vnos podatkov ter nuditi uporabniku intuitiven vmesnik pri izvajanju poslovnih procesov. Na predstavitvenem nivoju se izvaja le osnovno preverjanje poslovnih pravil.

- **Poslovni nivo**. Namenjen je izvajanju poslovnih procesov in aktivnosti. Prav tako je odgovoren za uveljavljanje poslovnih pravil in omejitev. Poslovni nivo je lahko razdeljen na več slojev. Običajno ga sestavljata aplikacijski sloj (odgovoren za potek procesov, integracije...) in poslovni sloj (osredotoča se na poslovna pravila in entitete).

- **Podatkovni nivo**. Namenjen je zanesljivemu, trajnemu hranjenju podatkov. Poleg tega s svojo zbirko orodij omogoča tudi napredne statistične obdelave in analize.

Med življenjskim ciklom aplikacije omogoča trinivojska arhitektura določene prednosti (v primerjavi z dvo- ali enonivojsko), kot so ponovna uporabnost, prožnost, obvladljivost, vzdrževalnost in nadgradljivost. Po drugi strani je zaradi večje razpršenosti fizičnih lokacij nivojev (glede na dvo- ali enonivojsko arhitekturo) več mrežnega prometa po mrežnih povezavah [5].

### *1.1.2 SOA*

Aplikacija je načrtovana na osnovah storitveno orientirane arhitekture (v nadaljevanju SOA). Storitev je samozadostna enota funkcionalnosti. Po definiciji je storitev operacija, ki je lahko samostojno pozvana. Vsaka storitev implementira vsaj eno operacijo (npr. pridobitev seznama letalskih letov, rezervacija izbranega leta). Za vsako od storitev velja, da uporabniki storitve poznajo samo shemo sporočil in spisek operacij. Obnašanje storitve definirano s storitveno politiko [7].

Takšen pristop prinaša možnost:

- ohlapno povezanih servisov, v katerih je konkretno implementacijo storitve relativno enostavno nadomestiti z drugo implementacijo;

- kreiranja novih servisov prek orkestracije in ponovne uporabe obstoječih storitev;
- nudenja storitve zunanjim poslovnim partnerjem;
- prihodnjih nadgradenj brez vpliva na obstoječe odjemalce.

Poleg zgoraj omenjenih »tehničnih« prednosti vodi SOA k načrtovanju sistemov, ki so prilagojeni poslovnim potrebam in procesom in ne obratno. To pomeni, da s storitvijo modeliramo poslovni proces. Poslovne aktivnosti in akcije pa so modelirane s storitvenim vmesnikom (pogodbo), sporočili in operacijami [6].

### *1.1.3 Povezava z zunanjimi sistemi*

Aplikacija je povezana z zunanjimi sistemi. Z nekaterimi poteka sinhrona komunikacija z ostalimi pa asinhrona komunikacija. Asinhrona komunikacija poteka brez vpletenosti uporabnika. Sinhrona komunikacija se izvede kot del določenih operacij, za katere velja, da jih sproži uporabnik. V tem primeru operacija čaka na uspešen konec komunikacije z zunanjim sistemom, preden lahko sama zaključi z izvedbo.

### *1.1.4 Uporabljene tehnologije*

Predstavitveni nivo je implementiran na platformi Microsoft .NET 3.5, v tehnologiji Windows Forms. Aplikacijski nivo je implementiran na platformi Microsoft .NET 3.5. Podatkovni nivo je implementiran na platformi Microsoft SQL Server 2008. Razvojno okolje za .NET je Microsoft Visual Studio 2008. Razvojno okolje za MS SQL sta Microsoft Visual Studio 2008 in MS SQL Manager 2008.

## 1.2 Predstavitev problematike

### *1.2.1 Postopen upad odzivnosti*

V nekaj samostojnih in poslovno nepovezanih organizacijah uporabljajo Aplikacijo že 7 let. V eni izmed organizacij (v nadaljevanju Organizacija), ki je po številu uporabnikov, odjemalcev in transakcij največja, uporabniki ugotavljajo, da se od prve nameščene različice dalje odzivnost Aplikacije postopoma zmanjšuje. Ugotavljajo tudi, da je stopnja poslabšanja različna glede na operacijo, ki jo uporabnik sproži. Pri nekaterih operacijah je odzivnost že moteča. Izpostavljajo tudi, da odzivnost posamezne operacije dnevno ni vedno enaka.

Naknadne meritve so potrdile, da je pri določenih uporabniških operacijah odzivnost pod mejo, pri kateri je zadovoljstvo večine uporabnikov še sprejemljivo, zato je vložitev napora v izboljšanje odzivnosti, vsaj kritičnih oziroma izpostavljenih operacijah, nujnost.

### *1.2.2 Dolgoročno spremljanje odzivnosti*

Formalen postopek izpostavljanja nezadovoljstva nad odzivnostjo Aplikacije uporabnikom ni na voljo. Postopen upad odzivnosti je bil izpostavljen kot resen problem šele po večjem številu pritožb nezadovoljnih uporabnikov vodstvu Organizacije. Posamezne pritožbe so sicer prišle do vzdrževalca Aplikacije vendar se zaradi nerazpoložljivih meritev, spremenljive odzivnosti Aplikacije glede na del dneva in dvoma v objektivnost uporabnikovih opažanj, obseg in resnost problema nista ustrezno ocenjevala.

Vzpostavitev samodejnega merjenja, sprotnega in dolgoročnega preverjanja meritev lahko zagotovita hitro zaznavo upadanja odzivnosti (na splošno ali v posameznih operacijah) in pravočasno naslavljanje pojavljajoče se težave.

### *1.2.3 Dogodki čezmerno nizke odzivnosti*

Poleg postopnega upada odzivnosti, uporabniki kot moteč dejavnik navajajo sicer redko (npr. enkrat ali dvakrat na mesec) vendar čezmerno nizko odzivnost v trajanju tudi do pol ure (izjemoma več). Posledično je onemogočen normalen potek dela in produktivnosti uporabnikov se zmanjša.

Ker je po oceni vodstva Organizacije v času trajanja teh dogodkov produktivnost uporabnikov lahko nezanemarljivo zmanjšana, je odprava ali vsaj zmanjšanje tega problema tudi poslovno pomembna odločitev.

## 1.3 Opredelitev ciljev

Zgodovinskih podatkov o odzivnosti Aplikacije nimamo na voljo, zato je ocene uporabnikov glede slabšanja odzivnosti težko neposredno preverjati. Glede na določene dejavnike, ki so običajni v življenjskem ciklu večine aplikacij (rast kompleksnosti izvorne kode in količine podatkov) menimo, da so mnenja uporabnikov verodostojna in da držijo. Dopuščamo, da so k zmanjšanju lahko prispevali tudi drugi vzroki. V vsakem primeru pa je treba take pritožbe uporabnikov ustrezno nasloviti.

Da bolje razumemo, katere operacije so za uporabnike najbolj moteče, smo uporabnike pozvali, da identificirajo (s seznamom) tiste, katerih trajanje je zanje moteče. Za vse operacije s tega

seznama (v nadaljevanju uporabniški seznam) so uporabniki navedli tudi zadovoljiv povprečni čas. Ti časi so pomenili orientacijo za postavljanje ciljev reševanja problema. Vrstni red v seznamu je bil naključen zato je bilo treba prioritetni vrstni red reševanja še določiti.

Na splošno je postavljanje ciljev pri pohitritvah v vnaprej določenih maksimalnih (ali povprečnih) odzivnih časov operacij lahko sporno. Še posebej, če jih postavljajo uporabniki, ki se praviloma ne zavedajo kompleksnosti in obsega, izvajanega v sklopu posamezne operacije. Povprečne odzivne čase glede na operacijo, ki so jih uporabniki označili kot zadovoljive (v uporabniškem seznamu), smo zato opredelili kot smernice in prioritete procesa izboljševanja odzivnosti. Proces izboljševanja ni bil zastavljen kot enkraten napor, ampak bo potekal vzporedno z drugimi obveznimi aktivnostmi razvoja in vzdrževanja Aplikacije.

Hkrati je bil z odgovornimi iz Organizacije sprejet dogovor o spremljanju odzivnosti kot trajnem procesu nadaljnjega življenjskega cikla Aplikacije. Za zagotovitev ustreznega spremljanja je treba zagotoviti dolgoročno samodejno izvajanje ustreznih meritev, njihovo obdelavo in nazoren prikaz.

Cilji reševanja problema občasnih čezmerno nizkih odzivnosti so v preprečitvi takih situacij v obsegu mogočega. Če to ni mogoče, naj bo zaznavanje takega stanja in ukrepanje odgovornih kar se da hitro.

Pri iskanju ustreznih rešitev na izpostavljene probleme smo vedno upoštevali tudi morebiten strošek rešitve.

## 1.4 Metodologija

### *1.4.1 Pristop k reševanju postopnega upada odzivnosti*

Glede na priporočene prakse s področja izboljšave odzivnosti [3] smo se odločili, da bomo:

- pohitrili nekaj tistih podoperacij, ki so skupne čim večjemu številu operacij na splošno (predvidoma tudi večini z uporabniškega seznama);
- nato izboljšali odzivnost konkretnih posamičnih operacij z uporabniškega seznama po vrstnem redu, ki bi okvirno izhajal iz razlike med izmerjenim (povprečnim) in zadovoljivim (povprečnim) časom izvedbe operacije.

### 1.4.2 Dolgoročno spremljanje odzivnosti

Ocenili smo, da bomo sprotno spremljanje odzivnosti Aplikacije omogočili z merjenjem trajanja izvedb vnaprej določenih reprezentativnih operacij. Meritve je treba ustrezno grafično ponazoriti tako, da je razviden trend ali sprememba odzivnosti. Smiselno je tudi implementirati možnost samodejnega obveščanja odgovornih, če izmerjena trajanja izvedb operacij presežejo vnaprej določene meje.

### 1.4.3 Pristop k reševanju dogodkov čezmerno nizke odzivnosti

Na podlagi analize dogodkov čezmerno nizke odzivnosti je bil kot vzrok identificiran znižan odzivni čas podatkovnega strežnika. Posledica tega je bistveno daljše trajanje izvedb rednih operacij uporabnikov. Vzrok znižane odzivnosti podatkovnega strežnika je povečana obremenitev nekaterih njegovih virov, ki je posledica sočasnega izvajanja dveh ali več nerednih operacij (npr. izdelava obsežnejšega poročila, mesečni obračun ...). Predvsem takih, ki omogočajo delovne parametre, ki so lahko zelo zahtevni do virov podatkovnega strežnika.

Preveriti je treba možnosti bolj optimalne izvedbe nerednih operacij, da se zagotovi neproblematično obremenitev virov podatkovnega strežnika. Če to ni mogoče ali samo delno uspešno, je treba implementirati samodejno spremljanje dogajanja na podatkovnem strežniku. Če je v okviru samodejnega spremljanja zaznan pojav znižanja odzivnosti pod vnaprej določeno mejo, se samodejno izvede takojšnje obveščanje odgovornih. Če je samodejno ukrepanje mogoče, naj se izvede.

## 1.5 Struktura naloge

V prvem poglavju smo opisali arhitekturne elemente, uporabljene tehnologije in okolje Aplikacije. Izpostavili smo problematiko, razčlenjeno na tri probleme. Omenili smo omejitve in predpostavke pri reševanju problematike ter obrazložili uporabljene metodologije (za vsak problem posebej).

V drugem poglavju bomo podrobneje opisali problem postopnega upada odzivnosti z navedbo vzrokov, natančnejšo opredelitvijo odzivnosti, razlago postopka merjenja in orisom smeri reševanja. V tretjem poglavju bomo opisali problem nespremljanja odzivnosti. Podali bomo implikacije in način naslavljanja problema. V poglavju 4 bomo obravnavali dogodke čezmerno nizke odzivnosti in ugotovili, kaj so najpogostejši vzroki, ter navedli izvedene ukrepe. V zaključku bomo na kratko povzeli problematiko, izpostavili bistvene dele naloge in podali naučeno.

# Poglavje 2 Reševanje postopnega upada odzivnosti

V poglavju 1.3 smo omenili, da je postopno upadanje odzivnosti do neke mere pričakovano zaradi dejavnikov, ki so običajni v življenjskem ciklu večine aplikacij. V tem poglavju bomo pogledali, kateri so ti dejavniki, kako vplivajo na odzivnost in kakšen je njihov obseg v Aplikaciji. Omenili bomo tudi morebiten vpliv drugih dejavnikov in opredelili natančneje pomen besede odzivnost. V nadaljevanju bomo opisali implementacijo merjenja, navedli nekaj primerov pohitritve skupnih podoperacij in pristop k pohitritvi operacij z uporabniškega seznama.

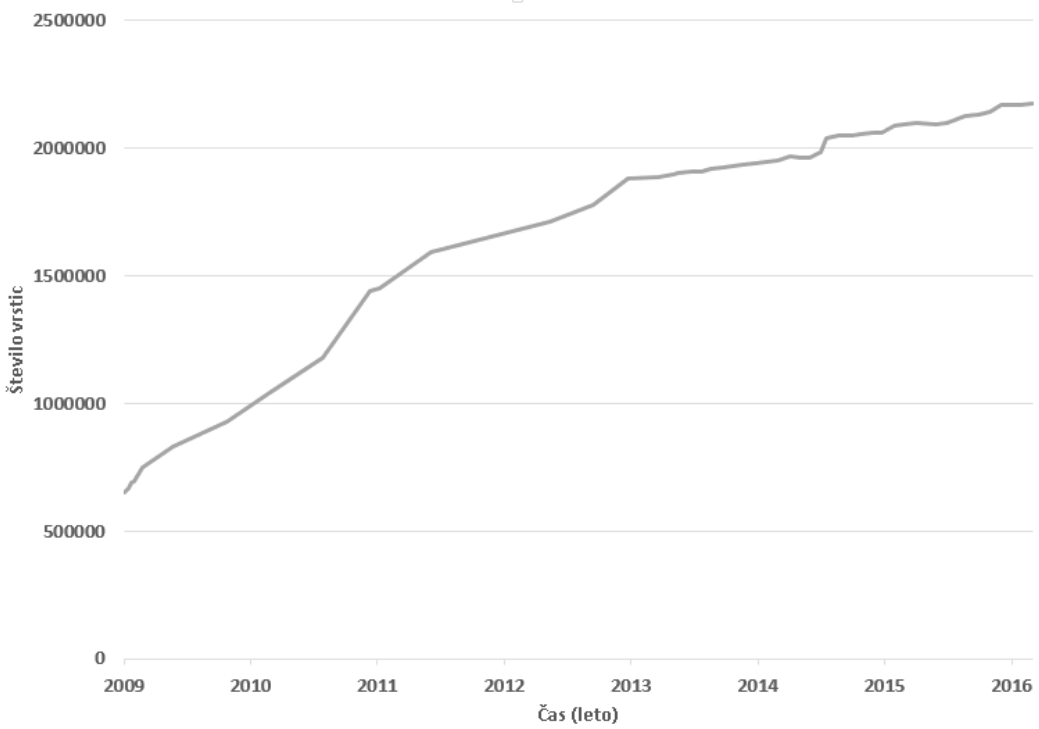
## 2.1 Vzroki postopnega upada odzivnosti

### *2.1.1 Kompleksnost izvorne kode*

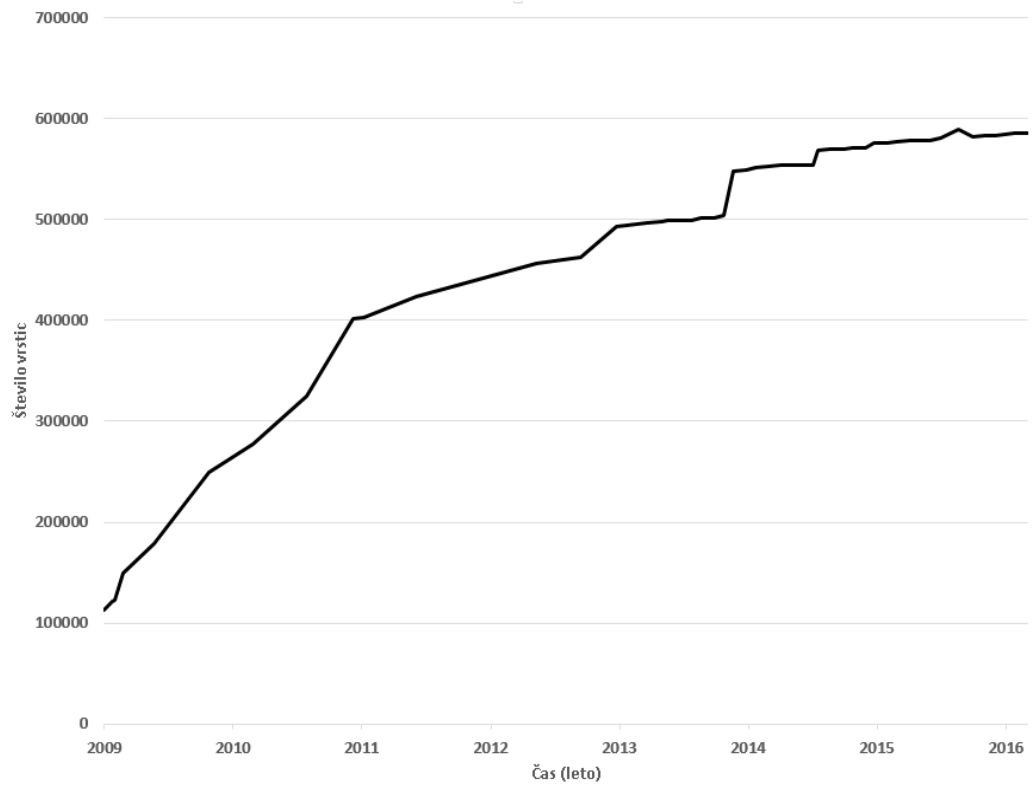
Aplikacija se je od prve različice v toku posodobitev oziroma novih različic zelo spremenila glede števila novih in obsega obstoječih funkcionalnosti (operacij, ki jih Organizacija želi imeti informacijsko podprte). Posledično narašča kompleksnost Aplikacije.

Na sliki 2.1 je prikazan časovni potek skupnega števila vrstic izvorne kode v jeziku C# (od prve različice dalje), ki je v Aplikaciji prevladujoč programski jezik. Ta koda se izvaja na odjemalcih in aplikacijskem strežniku. Število vrstic izvorne kode se od prve različice dalje povečuje iz dveh razlogov. Prvi je prek posegov dodajanja novih operacij in drugi prek posegov razširjanja funkcionalnosti obstoječih operacij. Predvsem posegi slednjega tipa praviloma povzročajo daljšo izvedbo operacij (predmet razširjanja). Po mnenju nekaterih članov razvojne ekipe Aplikacije, sta bili vsaj dve tretjini novih vrstic izvorne kode (od prve različice dalje) dodani na račun razširitev obstoječih operacij. Postopno nižanje odzivnosti oziroma daljši čas izvedbe tistega dela operacij, ki se izvaja na odjemalcu in aplikacijskem strežniku je zato pričakovan.

Na sliki 2.2 je prikazan časovni potek skupnega števila vrstic izvorne kode v skriptnem jeziku Transact-SQL (T-SQL) (od prve različice dalje), ki se izvaja na podatkovnem strežniku. Tudi za število vrstic v skriptnem jeziku T-SQL velja, da se od prve različice dalje povečuje iz že navedenih razlogov. Postopno nižanje odzivnosti oz. daljši čas izvedbe operacij na podatkovnem strežniku je zato pričakovan.



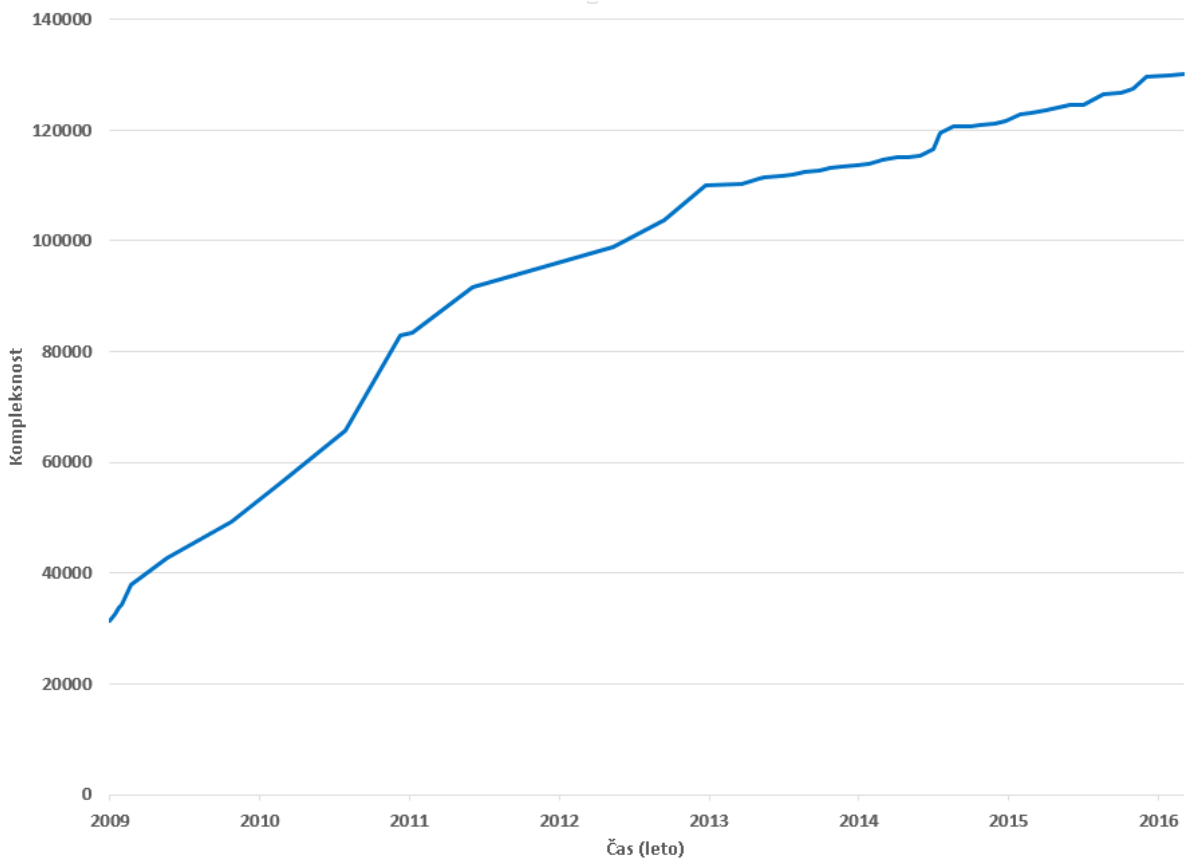

Slika 2.1: Količina kode v jeziku C#




Slika 2.2: Količina kode v skriptnem jeziku T-SQL

Ciklomatična kompleksnost je metrika za izvorno kodo, ki nakazuje kompleksnost. Metrika je merilo kvantitete števila linearno neodvisnih poti skozi izvorno kodo. Metriko je razvil Thomas J. McCabe, Sr. leta 1976 [4]. Npr. če izvorna koda nima ukazov kontrole toka (pogojni stavki ...), potem je kompleksnost 1, saj obstaja samo ena pot skozi kodo. Če je v kodi en pogojni IF stavek potem obstajata dve poti. Prva, kjer je pogoj izpolnjen, in druga, kjer pogoj ni izpolnjen. Kompleksnost je v tem primeru 2. Ciklomatična kompleksnost se lahko izračuna za posamezne funkcije, module ali razrede v programu.

Pričakovano je, da bo z naraščanjem števila vrstic kode naraščala tudi kompleksnost Aplikacije. Na sliki 2.3 je prikazan časovni potek (od prve različice dalje) ciklomatične kompleksnosti Aplikacije (upoštevana je izvorna koda v programskem jeziku C#).



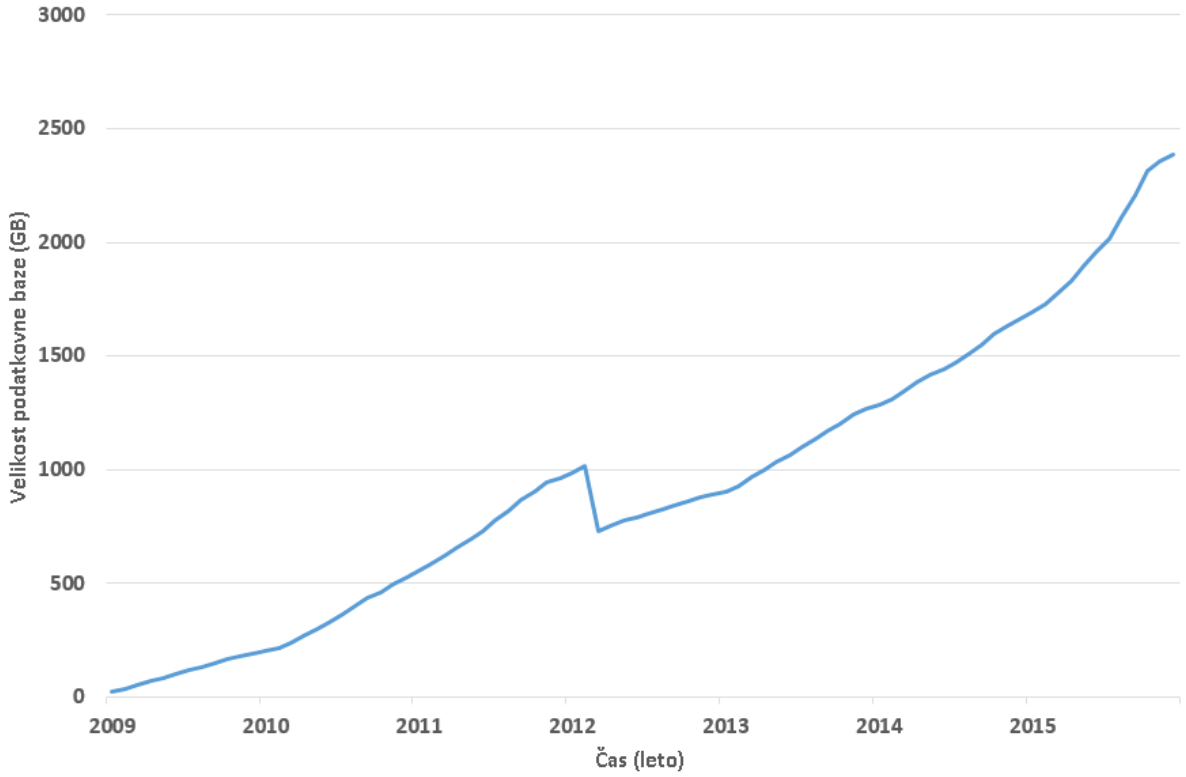

Slika 2.3 : Časovni potek ciklomatične kompleksnosti Aplikacije

### 2.1.2 Količina podatkov

Do sredine leta 2012 so bili podatki razdeljeni na dve podatkovni bazi. Določen del podatkov (npr. šifranti) se je iz praktičnih razlogov nahajal v obeh bazah (podvajanje). Razlog za začetno porazdelitev podatkov v dve bazi je bil performančne narave (večja stopnja paralelizma).

Vendar je taka razdelitev hkrati omejevala možnosti interne optimizacije strežnika SQL (poznane kot SQL Server Query Optimization*)* pri izvajanju poizvedb (npr. poročila, statistike ...), ki so vključevale podatke iz obeh baz. Z naraščanjem števila podatkov je omejitev povzročala naraščajoče performančne težave. Težave so se lahko rešile le z združitvijo obeh baz. Velikost združene baze je bila zaradi ukinitve (po novem nepotrebnega) podvajanja manjša kot skupna velikost obeh baz. Padec v velikosti je razviden s slike 2.4.

Naraščanje količine podatkov je običajno za vse transakcijske sisteme. Praviloma se na to vezane težave lahko rešuje z arhiviranjem dela podatkov, ki niso več aktualni za tekoče poslovanje. Arhivirani podatki so glede na poslovne potrebe vseeno lahko dostopni vendar pod drugačnimi (slabšimi) pogoji vsaj kar zadeva dostopnosti. V Aplikaciji je obseg podatkov, ki jih je mogoče arhivirati, zanemarljiv. Iz poslovnih razlogov večina podatkov vedno velja za aktualne. Posledično sčasoma (in z nezanemarljivo hitrostjo) narašča količina podatkov in povezanih podatkovnih struktur v podatkovni bazi. Slika 2.4 prikazuje dinamiko tega naraščanja v Organizaciji.




Slika 2.4: Časovni potek velikosti podatkovne baze

### 2.1.3 Drugi vzroki postopnega upada odzivnosti

Ker nimamo celovitega pregleda nad dogajanjem (npr. spremembami) v infrastrukturi Organizacije, težje ocenjujemo pojavnost vzrokov izven Aplikacije, ki bi lahko povzročili postopen upad odzivnosti ali čezmerno nizko odzivnost. Zagotovo pojavov takih vzrokov ne moremo povsem izključiti. O tem priča tudi primer, ko so v manjši organizaciji sami identificirali in odpravili tak vzrok. V organizaciji so opazili postopen porast uporabe spletnih multimedijskih vsebin zaposlenih. Uporabnikom so zato preprečili dostop do določenih multimedijskih vsebin (npr. spletna aplikacija YouTube …). O slabi odzivnosti Aplikacije, blokiranju vsebin in posledično izboljšani odzivnosti smo bili obveščeni naknadno. Ker nismo imeli implementiranih ustreznih meritev, izboljšanja odzivnosti nismo mogli oceniti. Na podlagi tega primera so se tudi v Organizaciji odločili, da bodo uporabnikom onemogočili dostop do določenih multimedijskih vsebin.

Za postopno poslabšanje odzivnosti lahko poleg že navedenih vzrokov obstajajo tudi drugi, ki vključujejo (vendar niso omejeni na):

- souporabo infrastrukture in ostalih komponent zaradi drugih porabnikov;
- zmanjševanje zmogljivosti infrastrukture in drugih komponent.

Ocenjujemo, da je bodoče dogodke in napredujoči vpliv obstoječih faktorjev zmanjševanja odzivnosti mogoče zaznati z dolgoročnim spremljanjem odzivnosti.

## 2.2 Opredelitev odzivnosti

V diplomskem delu je opisano izboljšanje odzivnosti Aplikacije. Pomen besede odzivnost je v tem kontekstu specifičen zato je v nadaljevanju natančneje opredeljen.

Uporabnik praviloma s klikom na nek gumb v Aplikaciji v nekem trenutku zahteva izvedbo ene izmed možnih operacij. Dokler se operacija izvaja, je uporabniški vmesnik praviloma zaklenjen (angl. *application busy*), kar je mogoče razbrati tudi iz spremembe videza kazalca (angl. *cursor*) in rahlo posivljene ekranske maske. V času izvajanja so neodzivne vse grafične kontrole (gumb, zavihki …), zato uporabnik ne more nadaljevati z uporabo Aplikacije. Uporabnik lahko nadaljuje z uporabo šele, ko se uporabniški vmesnik odklene. To se zgodi, ko se izvedba operacije konča. Čas trajanja izvedbe operacije je čas odziva (odzivni čas) Aplikacije pri konkretni izvedbi operacije. Čakalni čas je tisti čas, ki se začne v trenutku, ko bi uporabnik lahko izvršil naslednjo akcijo, in konča v trenutku, ko lahko izvrši naslednjo akcijo.

Odzivni čas neke operacije je daljši kot čakalni čas. Če je čas izvedbe operacije 1 ms, potem velja, da lahko (človeški) uporabnik brez čakanja nadaljuje z uporabo aplikacije. Isto velja tudi v primeru, da je čas izvedbe 10 ms. Miselni (npr. ugotovitev, kaj je naslednji korak) in telesno-motorni proces (npr. premik miške, tipke …), ki je potreben za naslednjo uporabnikovo akcijo, je mnogo večji kot v primeru navedeni odzivni čas. Zdi se, da pohitritev izvedbe operacije s 5 ms na 1 ms pri predpostavki, da to ne vpliva na odzivnost ostalih sočasnih operacij pri drugih uporabnikih, uporabniku neposredno ne prinaša koristi.

Čas, ki ga uporabnik po zagonu neke operacije potrebuje, da konča miselni in telesno-motorni proces za izvedbo naslednje uporabniške akcije, imenujmo maksimalni nečakalni odzivni čas. V tem času uporabnik osvoji kontekst, ki je potreben za takojšnjo izvedbo naslednje akcije. Če uporabnik po tem času ne more takoj nadaljevati, ker je uporabniški vmesnik neodziven, bo v sklopu čakanja postopoma izgubljal osvojeni kontekst. Tega bo moral delno obnoviti v trenutku, ko se vmesnik odklene in Aplikacija spet omogoča nadaljnje delo. Velja, da maksimalni nečakalni odzivni čas ni konstanten, saj nanj vpliva kompleksnost naslednje akcije, trenutno razpoloženje, zbranost, trenutne sposobnosti uporabnika …

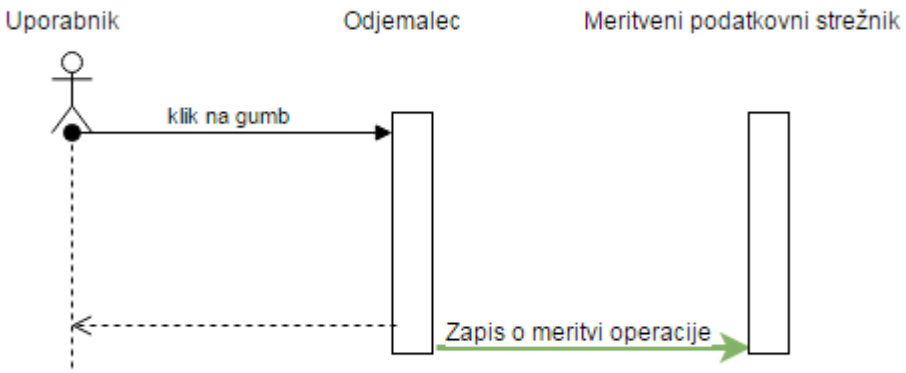
Uporabniki pri svojih dnevnih aktivnostih in obveznostih uporabljajo Aplikacijo s prekinitvami (v intervalih). Uporabnik izvede nekaj operacij na Aplikaciji in nato nadaljuje z zaporedjem opravil, ki niso za računalnikom. Slabša odzivnost pri zadnji izvedeni operaciji za uporabnika ni pomembna, saj naslednje opravilo uporabnika ne bo za računalnikom.

Običajen pristop k zmanjševanju izgube produktivnosti bi bil pohitritev operacij glede na vrstni red (operacij), ki izhaja iz ocene povprečne izgubljene produktivnosti. Ocena povprečne izgubljene produktivnosti je odvisna od maksimalnega nečakalnega odzivnega časa, ki ga je mogoče oceniti zaradi kompleksnosti in predvsem nepredvidljivosti parametrov, le na arbitraren način. Upoštevajoč, da je cilj reševanja težav predvsem zadovoljstvo uporabnikov, smo kot osnovo za reševanje vzeli uporabniški seznam.

## 2.3 Merjenje odzivnosti

### *2.3.1 Merjenje operacij*

Na podlagi uporabniških trditev je bilo najprej treba ugotoviti dejansko stanje, in sicer tistega dela, ki govori o moteči odzivnosti pri nekaterih operacijah (uporabniški seznam). Zato so bile vse operacije (ne glede na vnaprejšnjo dogovorjeno omejitev na uporabniški seznam), ki jih uporabnik lahko sproži (nekaj sto) opremljene z merilnimi mesti. Na sliki 2.5 je prikazan sekvenčni diagram poteka merjenja operacije.




Slika 2.5: Mejenje operacije

Zbirani podatki vključujejo:

- naziv računalnika odjemalca (angl. *hostname*);
- čas začetka operacije;
- naziv operacije (npr. naziv forme in gumba, ki jo sproži);
- čas trajanja operacije.

Ker so meritve vključevale vse izvedbe vseh operacij vseh uporabnikov je bilo mogoče preveriti kritičnost odzivnosti pri operacijah uporabniškega seznama in tudi začetno stanje pred začetkom procesa izboljševanja odzivnosti.

Na podlagi meritev v produkcijskem okolju smo opazili, da ima večina operacij nepričakovano veliko variabilnost trajanja izvedb. Pojav smo pripisali vrsti časovno spremenljivih notranjih parametrov (npr. število uporabnikov, količina obdelanih podatkov v sklopu operacije, količina prenesenih podatkov v sklopu operacije ...) in zunanjih parametrov (npr. trenutna prepustnost omrežja, trenutna obremenitev aplikacijskih strežnikov, trenutna obremenitev podatkovnega strežnika, trenutna prepustnost zunanjih sistemov ...), katerih vpliva nismo merili. Taka variabilnost je dopuščala možnost, da se občasno pojavljajo ozka grla v nekaterih delih infrastrukture zaradi trenutno izvajanih operacij Aplikacije in drugih aplikacij Organizacije (kjer kot skupna infrastruktura nastopa npr. mreža, diskovno polje). Ocenili smo, da bo podrobnejša analiza operacij z uporabniškega seznama usmerila pozornost v dele Aplikacije, katerih pohitritev bo posledično omilila tudi morebitne pojave ozkih infrastrukturnih grl.

Izvedba ene meritve traja v povprečju 1,5 ms (majhna varianca), kar je zanemarljivo glede na celotno trajanje povprečne operacije (1485 ms). Zato je z namenom trajnega spremljanja odzivnosti Aplikacije (več v poglavju 3) merjenje operacij trajno vklopljeno.

### 2.3.2 Merjenje odjemalčevih zahtev

V sklopu izvršitve ene operacije se izvede:

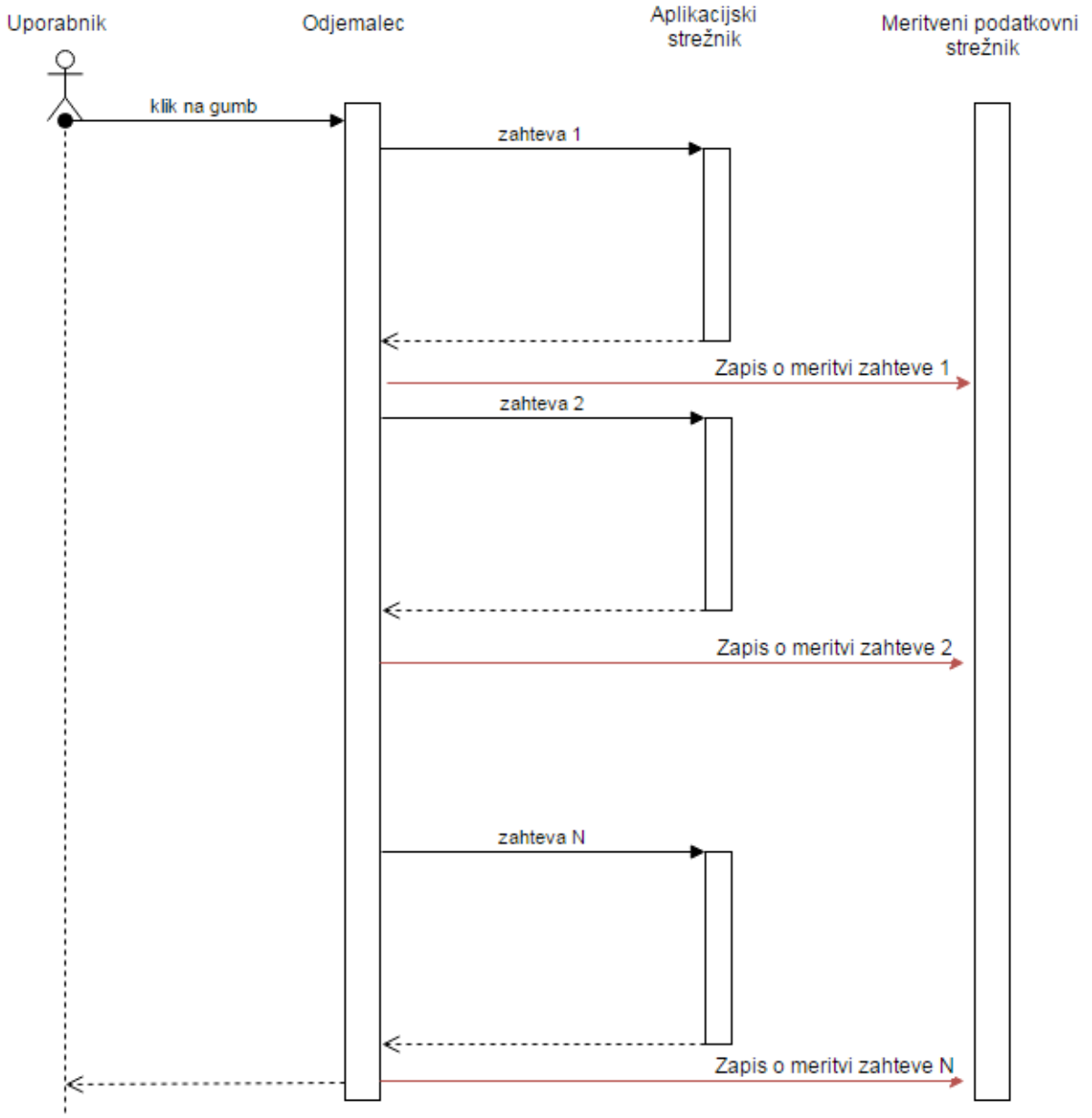
- del podoperacij na odjemalcu (uporabnikov računalnik);
- del podoperacij (na podlagi odjemalčevih zahtev) na aplikacijskem strežniku;
- del podoperacij (na podlagi zahtev odjemalca in aplikacijskega strežnika) na zunanjih sistemih;
- ostale podoperacije (na podlagi zahteve z aplikacijskega strežnika) na podatkovnem strežniku.

Šele, ko so vse podoperacije zaključene lahko uporabnik nadaljuje z uporabo Aplikacije. Za potrebe identifikacije spornih podoperacij so bila vsa mesta, na katerih odjemalec poda zahtevo na aplikacijski strežnik, opremljena z merilnimi mesti. Na sliki 2.6 je prikazan primer sekvenčnega diagrama izvedbe operacije in dodanih merilnih mest. Na sliki 2.6 ni prikazana morebitna zahteva, ki jo opravi odjemalec v zunanji sistem. Takih je malo, vendar so nezanemarljive po trajanju. Zato so tudi te opremljene z merilnimi mesti.

Zbirani podatki vključujejo:

- enolični identifikator izvedbe zahteve z odjemalca;
- naziv računalnika odjemalca (angl. *hostname*);
- čas podajanja zahteve;
- naziv zahteve (spletne storitve, metode);
- velikost zahtevka (v bajtih);
- velikost pripadajočega odgovora (v bajtih);
- (A) čas od podane zahteve do prejetega odgovora kot ga izmeri odjemalec;
- (B) čas izvedbe zahteve na aplikacijskem strežniku, kot ga izmeri aplikacijski strežnik;

- skupen čas izvedb (ene ali več) podatkovnih podoperacij v okviru zahteve, kot to izmeri aplikacijski strežnik.




Slika 2.6: Merjenje izvedbe zahtev na aplikacijski strežnik

Iz razlike med časoma (A) in (B) je mogoče ugotoviti, koliko časa je porabljenega skupaj za prenos zahtevka in pripadajočega odgovora po mreži (tehnično ni bilo mogoče zagotoviti posebej ugotavljanja časa za prenos zahtevka in posebej časa za prenos odgovora).

Izvedba ene meritve  traja v povprečju 1,5 ms (majhna varianca), kar je zanemarljivo glede na celotno trajanje povprečne odjemalčeve zahteve (331 ms). Zaradi pogostega izvajanje se tvori

obilica podatkov, zato je merjenje privzeto izklopljeno. Merjenje odjemalčevih zahtev se vklaplja po potrebi (tudi za več dni).

### 2.3.3 Merjenje izvedbe podoperacij na podatkovnem strežniku

Pričakovali smo, da bo primerno mesto pohitritve nekaterih operacij tudi na nivoju izvedbe podoperacije (oziroma skripte SQL) na podatkovni bazi. Da bi v primeru potrebe lahko merili izvedbe (s seznamom konkretnih vrednosti klicih parametrov) na podatkovnem strežniku, smo vsa mesta, na katerih se z aplikacijskega strežnika dajo tovrstne zahteve, opremili z merilnimi mesti.

Na sliki 2.7 je prikazan primer sekvenčnega diagrama izvedbe operacije in dodanih merilnih mest za merjenje izvedb na podatkovnem strežniku.

Zbirani podatki vključujejo:

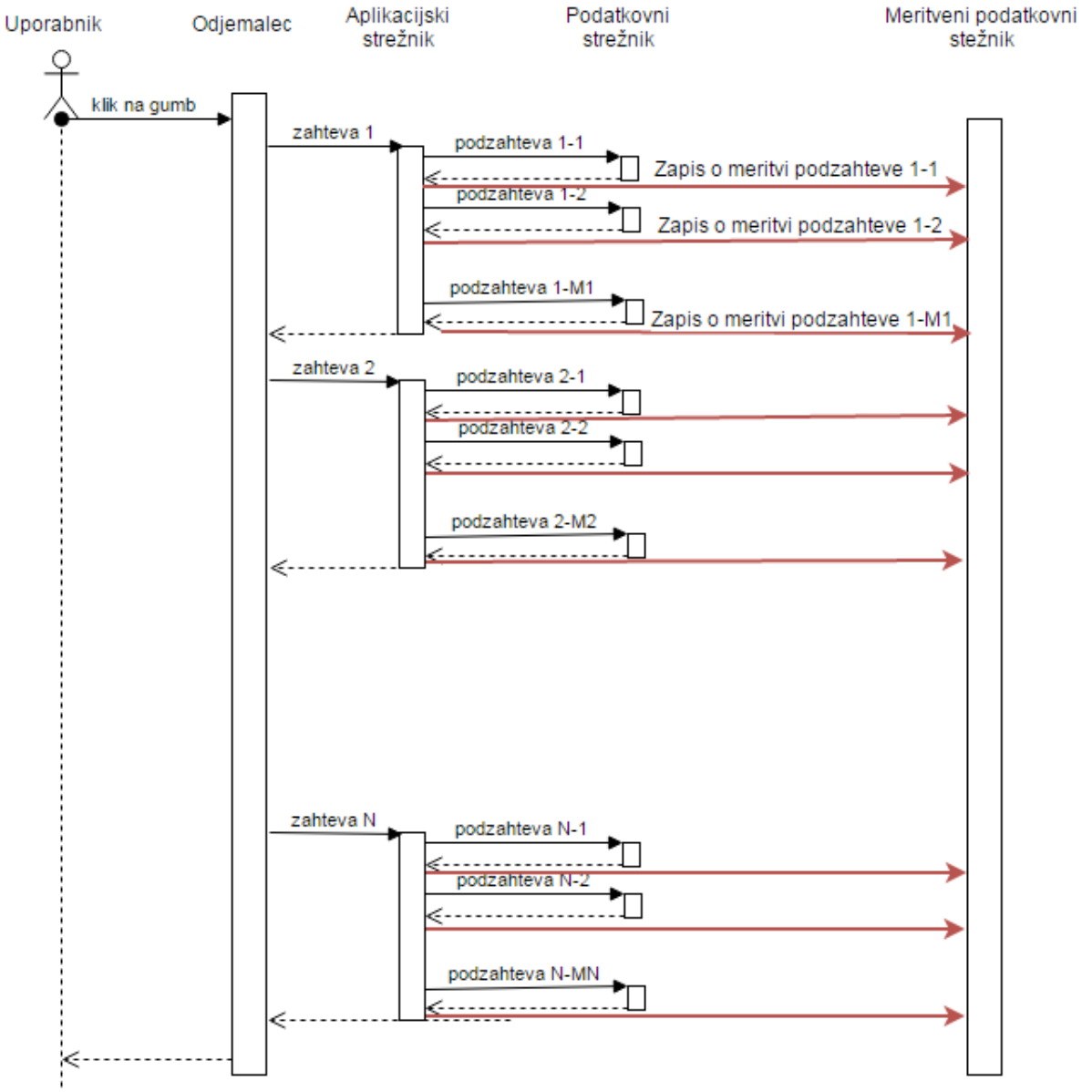
- enolični identifikator izvedbe zahteve z odjemalca;
- naziv skripte SQL, katere izvedbo zahteva aplikacijski strežnik;
- čas podajanja podzahteve;
- trajanje izvedbe podzahteve;
- nabor vhodnih parametrov skripte SQL s konkretnimi vrednostmi teh parametrov;
- število vrinjenih ali spremenjenih zapisov v bazi v okviru izvedbe.

S pomočjo podatka 'enolični identifikator izvedbe zahteve z odjemalca' je mogoče povezati te meritve z meritvami odjemalčevih zahtev. Tako je enostavno razbrati, v okviru katere odjemalčeve zahteve so se izvajale določene zahteve na podatkovni strežnik.

V primeru izboljševanja izvedbe neke operacije se najprej ugotovi katere odjemalčeve zahteve se izvedejo (v okviru izvedbe operacije). Nato se za vsako zahtevo preveri merjena podatka:

- čas izvedbe zahteve na aplikacijskem strežniku, kot ga izmeri aplikacijski strežnik;
- skupen čas izvedb (ene ali več) podatkovnih podoperacij v okviru zahteve, kot to izmeri aplikacijski strežnik.

Razlika med prvim in drugim pomeni čas izvajanja zgolj na aplikacijskem strežniku. Drugi pomeni čas izvedb samo na podatkovnem strežniku. V primeru, da je skupni čas na




Slika 2.7: Merjenje zahtev na podatkovni strežnik

podatkovnem strežniku velik, se preveri še meritve izvedb na podatkovni strežnik. Primerjava teh vrednosti omogoča uspešno identifikacijo mesta, na katerih se porabi največ časa za izvedbo odjemalčeve zahteve. Tisto mesto je potem predmet pohitritve.

Izvedba ene meritve traja v povprečju 1,5 ms (majhna varianca). Glede na celotno trajanje povprečne podatkovne operacije (13 ms) to ni zanemarljivo. Vendar velja, da se čas merjenja podatkovne operacije ne prišteva k izmerjenemu času izvedbe podatkovne operacije. Večina časa za izvajanje merjenja se namreč porabi za zapisovanje meritev v meritveno bazo, ko je meritev že končana. Vendar se zaradi pogostega izvajanja podatkovnih operacij (>300.000 /h)

zaradi merjenja zmanjšuje prepustnost aplikacijskega strežnika, ki mora te meritve pošiljati v hrambo na meritveni podatkovni strežnik, zato je merjenje privzeto izklopljeno. Po potrebi se vklaplja za krajše časovne intervale (npr. za uro ali manj).

## 2.4 Pohitritev skupnih podoperacij

### *2.4.1 Čas prenosa sporočil po omrežju*

Ena izmed skupnih podoperacij oziroma delov večine operacij je prenos sporočila v sklopu izvedbe zahteve. To je prenos podatkov po mreži med odjemalcem in aplikacijskim strežnikom. Kot sporočilo je v nadaljevanju opredeljen skup zahtevka in pripadajočega odgovora. Treba je poudariti, da čas prenosa ni odvisen od hitrosti Aplikacije, ampak od velikosti sporočila in trenutne prepustnosti mreže med odjemalcem in aplikacijskim strežnikom v tistem trenutku. Upoštevati je tudi treba, da imajo odjemalci, ki so v gručah razpršeni po različnih geografskih lokacijah, različne karakteristike povezav. Tudi odjemalci v isti gruči nimajo nujno povezav enakih karakteristik. Pri primerjavi merjenega časa prenosa sporočila med odjemalci je treba to upoštevati.

Na podlagi dvodnevnega merjenja (glej poglavje 2.3.2) v produkcijskem okolju Organizacije je nastalo približno 340.000 zapisov. Vsak zapis se nanaša na izvedbo ene zahteve na strežnik. Analiza meritev je pokazala, da traja prenos sporočila v povprečju 264 ms. V okviru večine operacij se izvede vsaj ena zahteva na aplikacijski strežnik (pri operacijah z uporabniškega seznama v povprečju tudi več). Zato smo ocenili, da je smiselno preveriti možnosti skrajšanja prenosa sporočil.

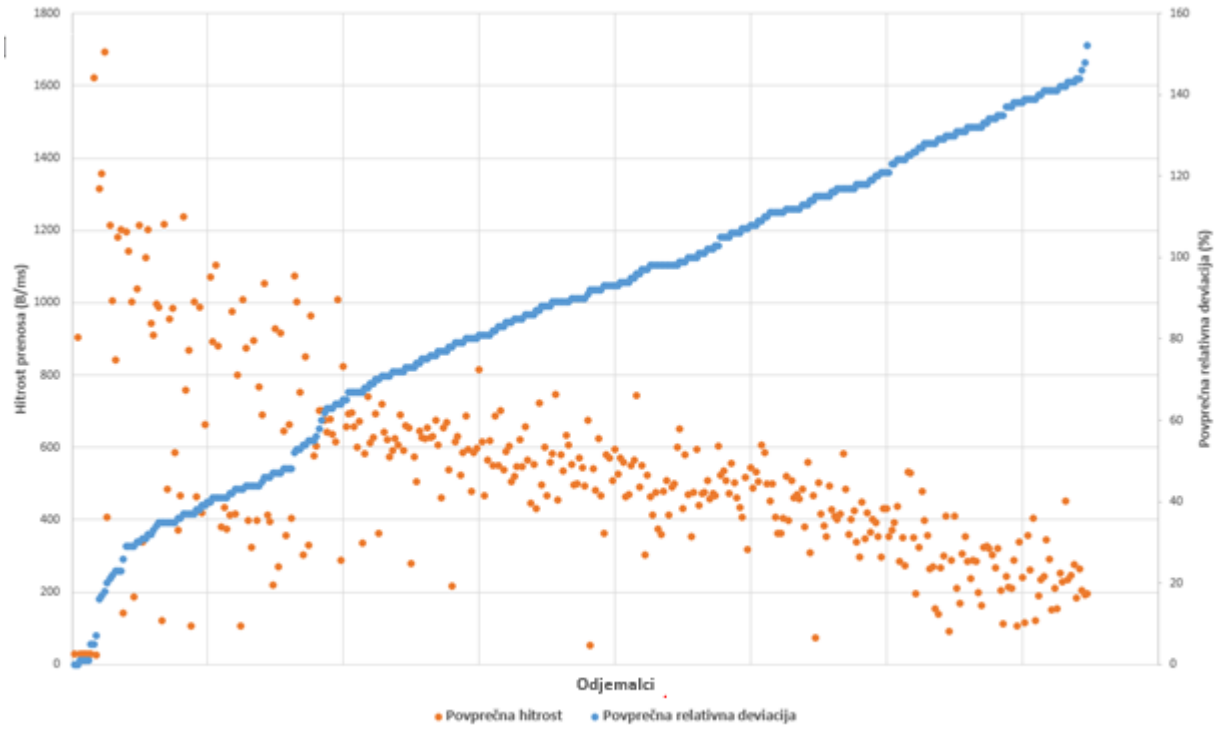
Ker so uporabniki navajali, da odzivnost Aplikacije podnevi ni enakomerna, smo posumili na dnevne epizode (časovne intervale) večje obremenitve omrežja, ki imajo za posledico daljše prenosne čase. Zato smo bili pri analizi meritev pozorni tudi na indikacije, ki bi sum potrdile.

Analizo smo omejili na najbolj množična (več kot tretjina) sporočila velikosti 15 do 16 KB. Zaradi množičnosti predstavljajo reprezentativen vzorec za ocenjevanje morebitne spremenljivosti hitrosti prenosa. Variabilnosti prenosnih časov neglede na odjemalca nismo preverjali, saj imajo odjemalci povezave različnih teoretičnih zmogljivosti, zato so različne vrednosti pričakovane. Zanimala pa nas je variabilnost znotraj meritev posameznega odjemalca (in kot že omenjeno za sporočila 15 do16 KB). Za vsakega odjemalca smo izračunali povprečno hitrost prenosa in povprečno relativno deviacijo slednje. Omenjeno deviacijo smo si izbrali kot kazalec spremenljivosti hitrosti prenosa pri odjemalcu. Na sliki 2.8 je razvidna povprečna relativna deviacija in povprečna hitrost prenosa – obe izračunani za vsakega odjemalca

posamično. Zato, da bi se morebitna korelacija lažje razbrala, so na grafu vrednosti prikazane po naraščajoči velikosti povprečne relativne deviacije. Na horizontalni osi bi torej morali biti navedeni vsi odjemalci, kar iz praktičnih razlogov ni mogoče.

Povprečna relativna deviacija se med odjemalci precej razlikuje. To je potrdilo sum, da posamezen odjemalec nima vedno (približno) enake hitrosti prenosa 15 do 16 KB sporočila (in verjetno tudi pri sporočilih ostalih velikosti). Odstopanja od povprečne hitrosti so namreč pri polovici odjemalcev v povprečju 100 % (glej desni del grafa).

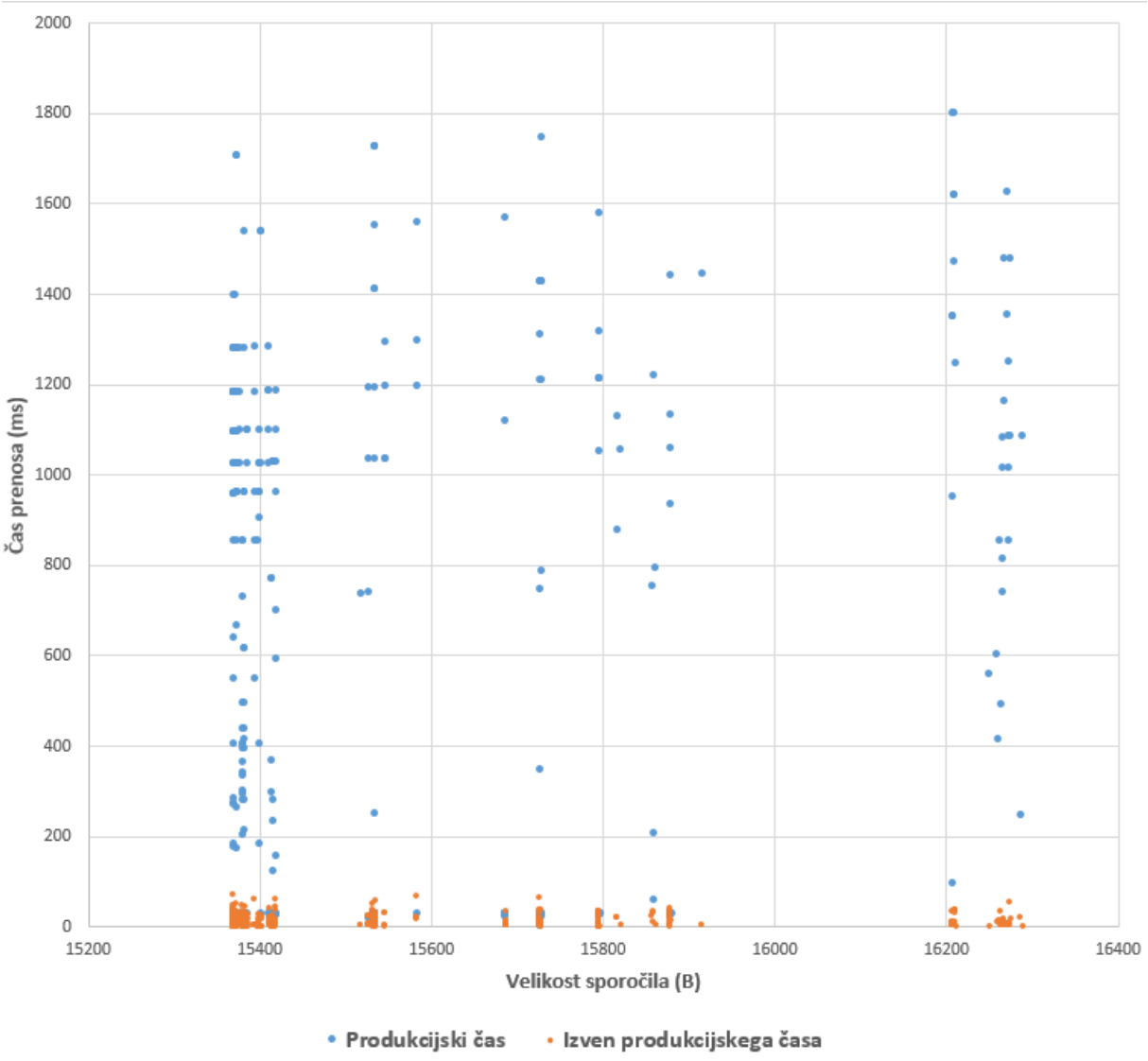
Da povečana variabilnost negativno vpliva na povprečno hitrost, je razvidno iz očitne korelacije med povprečno relativno deviacijo in povprečno hitrostjo odjemalca. Večja odjemalčeva povprečna relativna deviacija praviloma pomeni manjšo povprečno hitrost prenosa. Še najbolj očitno je to pri korelaciji, ki je večja od 60 %.




Slika 2.8: Povprečna relativna deviacija po odjemalcih in povprečna hitrost odjemalca pri sporočilih 15 do 16 KB

Da bi dodatno potrdili sum in ocenili obseg vpliva dnevne sočasne obremenitve omrežja zaradi ostalih porabnikov na povprečno hitrost prenosa, smo izvedli še dodaten test na izbranem odjemalcu izven delovnega časa. V okviru omenjenega testa se pošiljajo zahteve na aplikacijski strežnik. Testne zahteve so v številu in velikosti enake tistim, ki jih je odjemalec dal v okviru

produkcijskega delovanja med že omenjenim dvodnevnim merjenjem. Kriterij za izbiro odjemalca je bila sredinska vrednost (glede na ostale odjemalce) povprečne relativne deviacije. Na sliki 2.9 so prikazane meritve hitrosti prenosa izmerjene pri izvajanju testa izven produkcijskega časa. Na istem grafu so za primerjavo prikazane tudi meritve iz dvodnevnega produkcijskega merjenja. Če ponovimo: obe merjenji sta opravljeni na istem odjemalcu, z istim številom zahtev (403), istih velikosti (15 do 16 KB). Razlika je le v času izvajanja zahtev.




Slika 2.9: Hitrost prenosov izbranega odjemalca v produkcijskem času in izven produkcijskega časa

Rezultati so potrdili domnevo, da je vzrok manjše povprečne hitrosti prenosa med produkcijskim časom v souporabi povezave ostalih porabnikov. Morebitnih drugih vzrokov za razliko nismo identificirali.

Na podlagi ugotovljenega smo ocenili, da je smiselno preveriti možnosti, ki bi zmanjšale čas prenosov. Identificirali smo tri možne načine:

- kompresija zahtevkov in odgovorov (sporočil);
- blokada morebitnih drugih (neposlovnih) večjih porabnikov omrežnih povezav;
- izboljšava mrežne infrastrukture.

#### 2.4.1.1 Kompresija zahtevkov in odgovorov

Pri uvedbi kompresije je treba upoštevati naslednja dejstva:

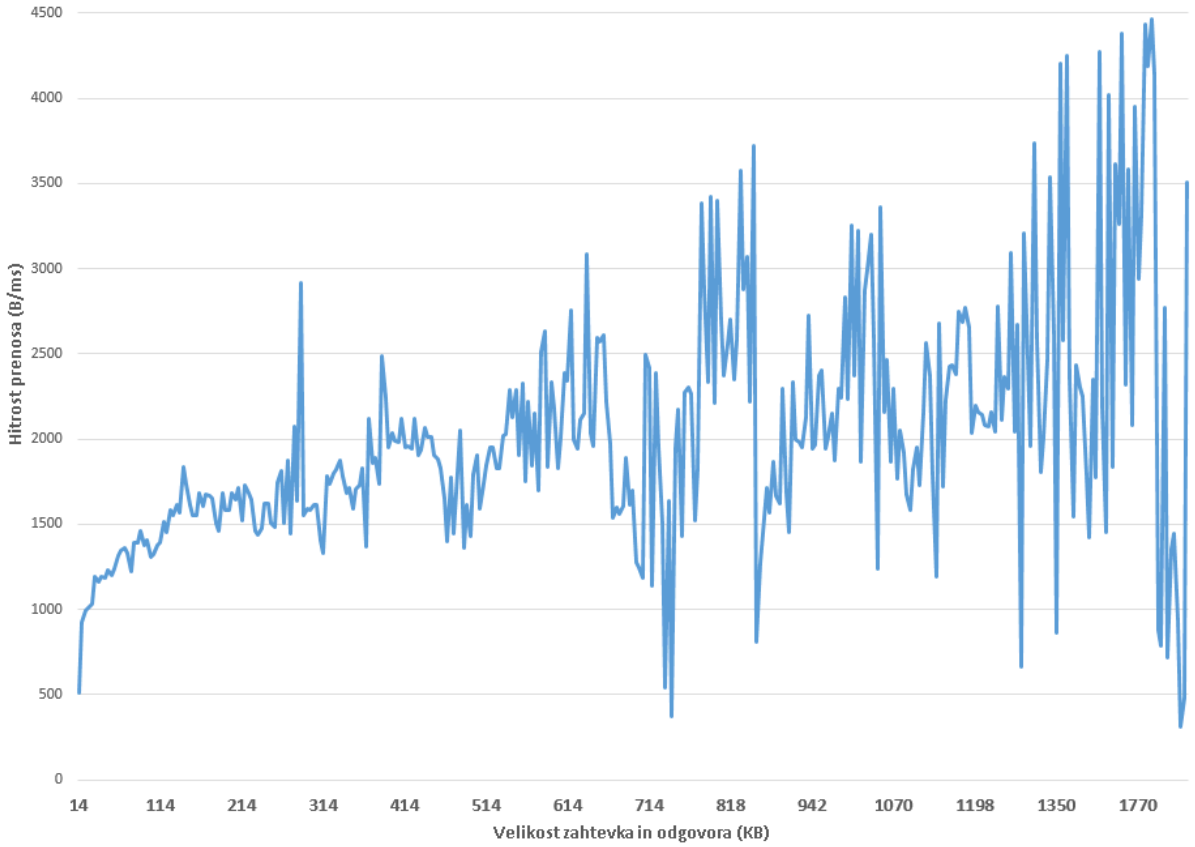
- Trajanje novih podoperacij kompresije in dekompresije se prišteva k celotnemu trajanju izvedbe zahteve.
- Dodatna obremenitev CPE zaradi kompresije in dekompresije lahko občutno vpliva na čas obdelave ostalih zahtev na aplikacijskem strežniku.
- Kompresija manjših sporočil bistveno ne zmanjšuje sporočila.

Upoštevajoč slednje dejstvo je bila implementacija kompresije narejena tako, da se kompresija izvede, samo če velikost sporočila presega neko vnaprej določeno nastavljivo mejo. Odločili smo se, da bomo optimalno mejo (oziroma dovolj dober približek) določili poskusno. To pomeni začetno nastavitev po vnaprej določenih časovnih intervalih (npr. dnevno) postopno nižati (in višati) in opazovati statistične vrednosti merjenih parametrov:

- nekompresirana velikost sporočila;
- kompresirana velikost sporočila;
- čas prenosa;
- čas za kompresije;
- čas za dekompresije;
- obremenjenost aplikacijskega strežnika (oziroma CPE).

Pri določitvi začetne vrednosti meje smo izhajali z grafov na slikah 2.10 in 2.11. Grafa sta nastala na podlagi že omenjenega dvodnevnega merjenja. Prvi prikazuje povprečno (ne glede

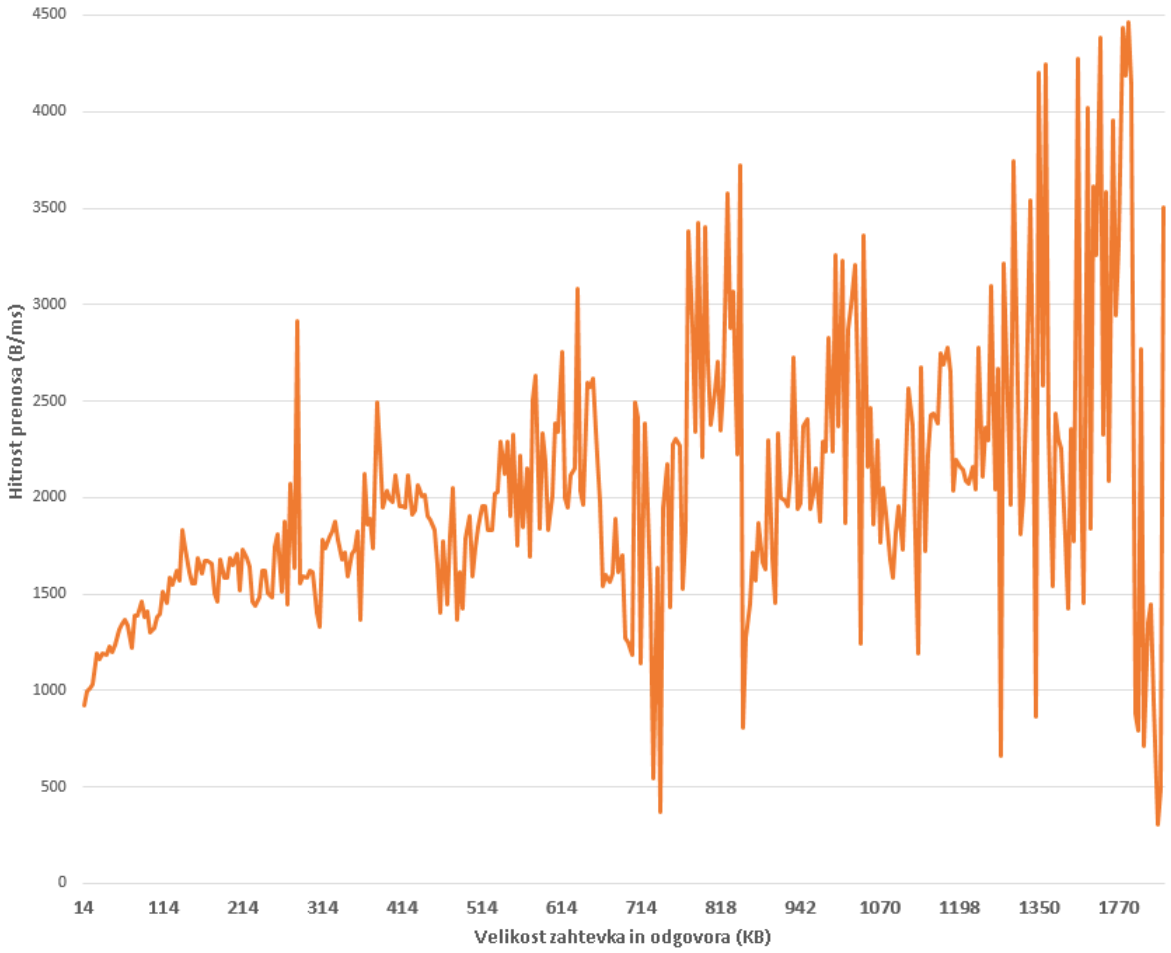
na odjemalca) hitrost prenosa glede na velikost sporočila. Drugi tudi prikazuje hitrost prenosa glede na velikost sporočila, vendar za samo enega izbranega odjemalca.




Slika 2.10: Povprečna hitrost prenosa glede na velikost sporočila pri odjemalcih

S slike 2.10 je razvidno, da se pri velikostih pod 50 KB z zmanjševanjem povprečne velikosti sporočila naglo zmanjšuje tudi povprečna hitrost prenosa. Pri ocenjenem povprečnem kompresijskem faktorju 2 smo se zato odločili, da bomo kot začetno mejo nastavili 100 KB.

Pri testiranju v razvojnem okolju je postopek kompresije v povprečju več kot prepolovil velikost sporočil (kompresijski faktor 2<). Vpliv dodatnega časa za izvedbo kompresije in dekompresije je bil v primerjavi z zmanjšanim časom prenosa zanemarljiv (pod 5% časa prenosa nekompresiranega sporočila). Zaradi relativno kratkega časa kompresije in dekompresije ter neopaznega povečanja uporabe CPE, smo ocenili, da bo vpliv na čas obdelave ostalih zahtev zanemarljiv. Ker sta v produkcijskem okolju zmogljivost aplikacijskega strežnika in zmogljivost mrežnih povezav drugačni kot v razvojnem okolju, smo želeli izmeriti učinkovitost neposredno v produkcijskem izvajanju Aplikacije.

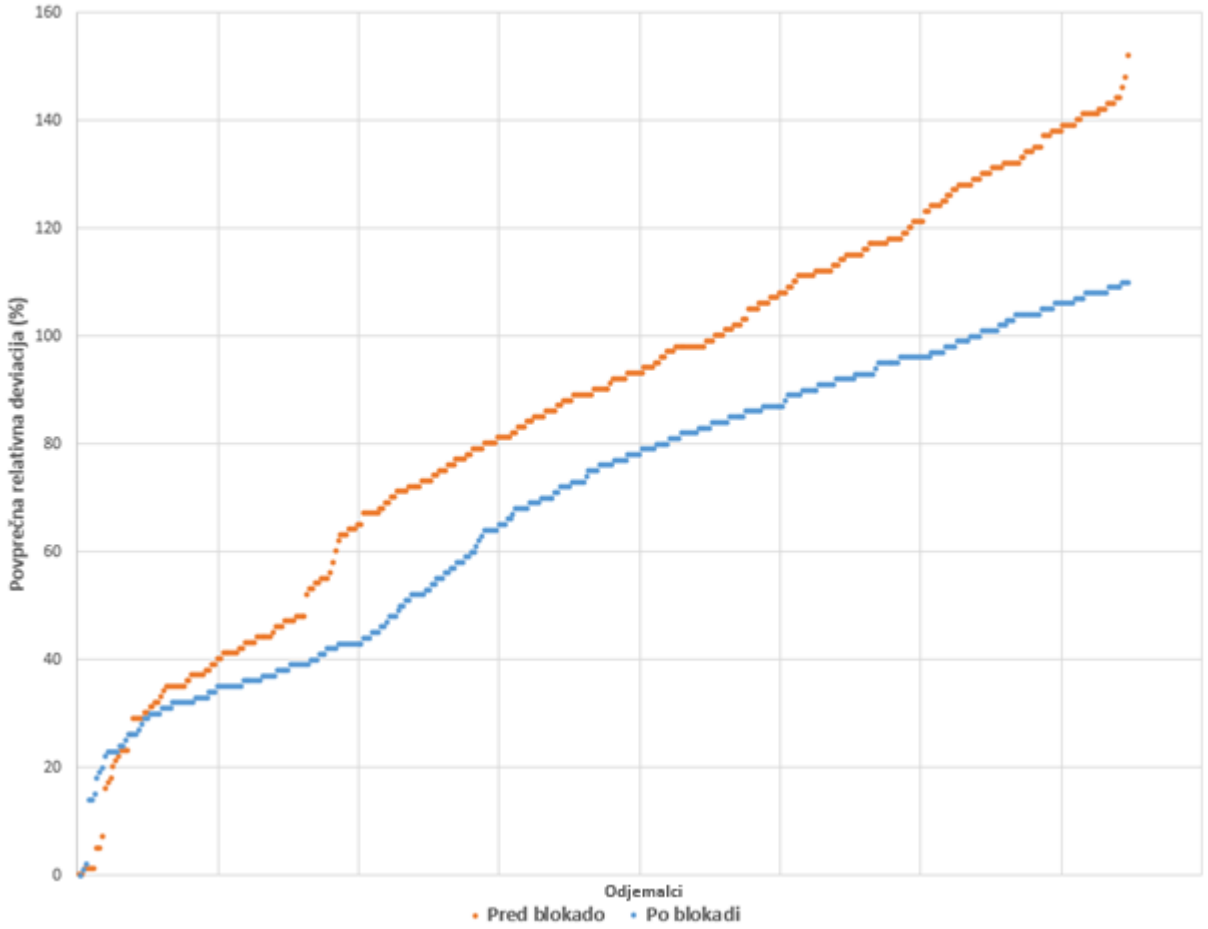



Slika 2.11: Hitrost prenosa glede na velikost sporočila za izbranega povprečnega odjemalca

Rezultati meritev uporabe kompresiranih sporočil v produkcijskem delovanju niso pokazali pričakovane izboljšave prenosnih časov glede na meritve nekompresiranih sporočil. Podrobnejša analiza je pokazala, da je bil kompresijski faktor v produkcijskem okolju zanemarljiv (včasih je celo povečal velikost sporočila). Na podlagi podrobnejše analize smo kot vzrok neučinkovitosti kompresije našli v postopku varnostnega šifriranja podatkov (oziroma vrstni red tega). V produkciji se zaradi občutljive narave prenašanih podatkov izvaja varnostno šifriranje sporočil. Izvaja ga komunikacijsko ogrodje WCF (angl. *Windows Communication Foundation*) na podlagi ustreznih konfiguracijskih nastavitev. Vtičnik WCF, v katerem smo implementirali kompresijo, se po vrstnem redu izvaja za postopkom šifriranja poročila. Želen rezultat šifriranja je efekt spreminjanja izvornih podatkov v podatke z visoko entropijo, kjer je slednja merilo nepredvidljivosti vsebine. Posledično so šifrirani podatki podobni naključnemu zaporedju bajtov. Vzorci v takih podatkih so precej manj verjetni. Kompresijski algoritmi učinkujejo najbolje, ko so v podatkih prisotni vzorci (ti so v okviru

kompresije predstavljeni z manj bajti). Navedeno je vzrok neučinkovitosti postopka kompresije. Krajše raziskovanje o tehničnih možnostih zamenjave vrstnega reda (še) ni bilo uspešno. Od morebitnega uspeha pri zamenjavi vrstnega reda si obetamo večji učinek pri pohitritvi prenosa zato še vedno poteka iskanje ustrezne rešitve. Med možnimi rešitvami je tudi prehod na novejšo različico WCF, ki ima vgrajeno možnost kompresirane komunikacije (brez potrebe za implementacijo kompresiranja prek vtičnika WCF).

#### 2.4.1.2 Blokada neposlovnih porabnikov mrežnih povezav

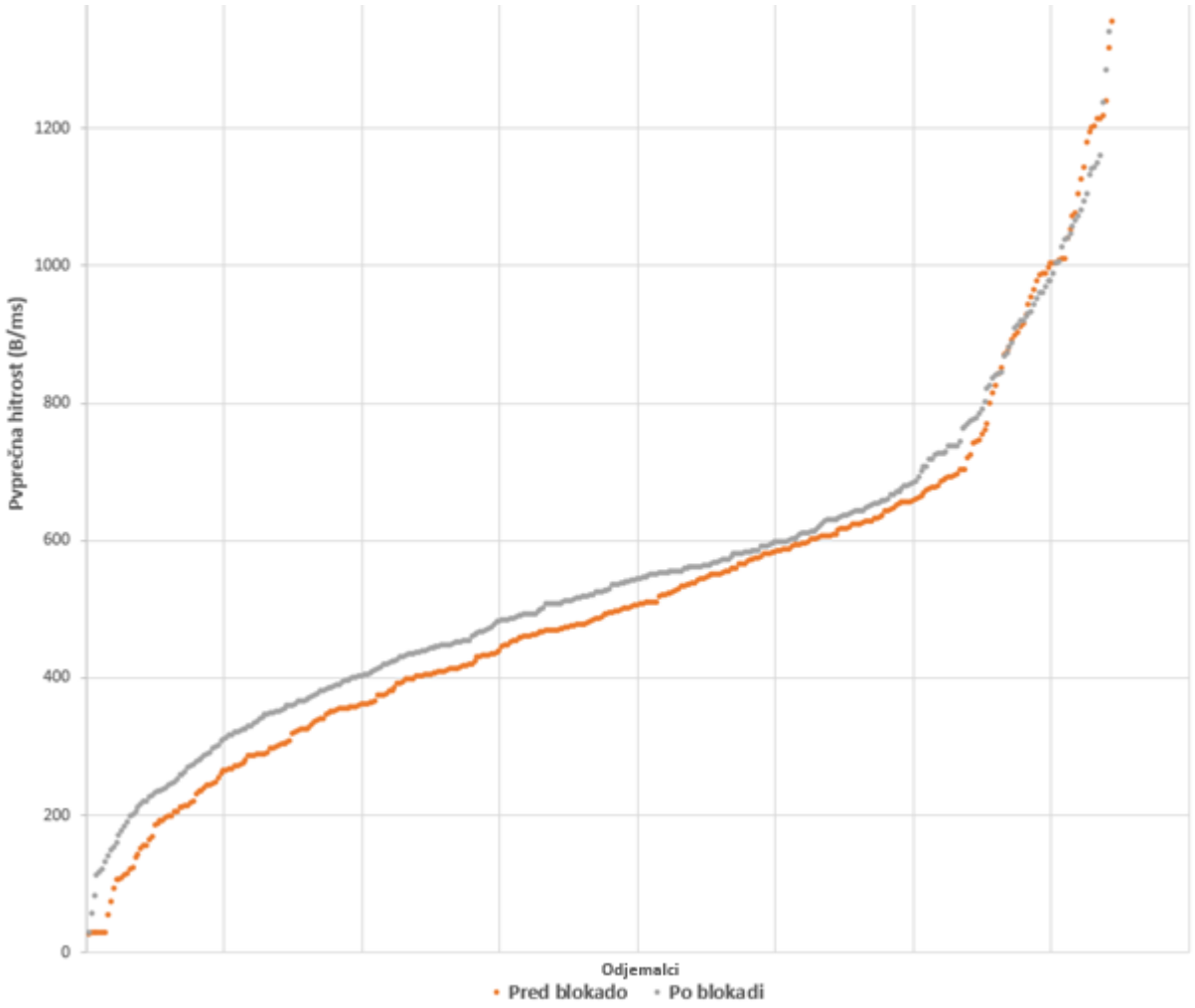



Slika 2.12: Velikostno urejena povprečna relativna deviacija po odjemalcih pred blokado in po njej

Predhodno je že bil omenjen primer, razlog in vtisi doprinosa blokade določenih spletnih multimedijskih vsebin v drugi organizaciji. Po tem zgledu se je tudi vodstvo Organizacije odločilo za blokado. Tokrat smo učinke izvedenega ukrepa merili. Na sliki 2.12 je razvidna povprečna relativna deviacija za vsakega odjemalca pred ukrepom in po njem.

Na sliki 2.13 je razvidna povprečna hitrost po odjemalcih (urejena po velikosti) pred ukrepom in po njem.

Odjemalci, ki so na sliki 2.12 na desni polovici grafa, so na sliki 2.13 pretežno v levi polovici grafa. Predvsem v levi (spodnji) četrtini grafa je razviden (relativno gledano) znaten pozitiven vpliv na povprečne hitrosti odjemalcev. To so ravni tisti, ki na nezadovoljstvo uporabnikov najbolj vplivajo. Glede na splošen trend naraščajoče uporabe multimedijskih vsebin na delovnem mestu je z blokado preprečeno bodoče dodatno postopno slabšanje prepustnosti povezav. Na podlagi navedenega menimo, da je ukrep učinkovit način izboljšave odzivnosti.




Slika 2.13: Velikostno urejena povprečna hitrost pri sporočilih 15 do 16 KB po odjemalcih pred blokado in po njej

#### 2.4.1.3 Izboljšava mrežne infrastrukture

S slike 2.9 je razvidna velika razlika časov prenosa v večernem (neprodukcijskem) in v dnevnem (produkcijskem) času. S slike 2.13 je razvidno, da noben od odjemalcev zaradi blokade nekaterih multimedijskih vsebin ni bil deležen tolikšne pohitritve, kot je očitna s slike 2.9. To pomeni, da podnevi neko drugo dogajanje (poleg multimedijske vsebine) na mreži

negativno vpliva na prenosno hitrost. Zato smo odgovornim v Organizaciji predlagali, naj počakajo na učinek uvedbe kompresije sporočil. Če kljub kompresiji dnevna hitrost ne bo primerljiva z nočno, potem naj z ustreznimi orodji spremljajo prepustnost in obremenjenost vseh segmentov uporabljanega omrežja in izvedejo ustrezne izboljšave infrastrukture.

### 2.4.2 Diskovno polje SSD

Na osnovi analize izvajanja na podatkovnem strežniku smo optimizirali izvedbo najbolj kritičnih podoperacij. Optimizacije podoperacij na programskem nivoju niso bile vedno mogoče, zato smo preverili, kateri viri podatkovnega strežnika predstavljajo ozka grla. Pri večini izvedb analiziranih kritičnih podoperacij je večji del časa potekalo branje in pisanje na diskovje. Vzdrževanje infrastrukture (vključujoč investicije) so izven domene razvijalca (in vzdrževalca) Aplikacije, zato smo predlog za nabavo zmogljivejšega diskovja predali odgovornim v Organizaciji. Na podlagi predloga so pridobili na preskušnjo in testiranje diskovje s tehnologijo SSD. Začasna zamenjava obstoječega diskovja Aplikacije z izposojenim diskovjem SSD in merjenjem vnaprej določenih kazalcev bi omogočala zelo zanesljive ocene o pohitritvi. Hkrati bi tudi z uporabniškega stališča lahko pridobili oceno (vtis) o izboljšavi. Zaradi časovno tehničnih razlogov je bilo odločeno za hitrejši test, ki sicer daje natančno primerjavo zmogljivosti obstoječega diskovja in diskovja SSD, vendar manj natančno na nivoju odzivnosti konkretnih uporabniških operacij.

Z namenom ocenjevanja morebitne pohitritve je bilo diskovje SSD dodano na testno okolje Aplikacije. Organizacija vzdržuje testno okolje za namen testiranja prihajajočih različic, raznim preverbam in poskušanju raznih scenarijev ter drugih morebitnih potreb. Do testnega okolja lahko dostopa le peščica izbranih uporabnikov (in vzdrževalec Aplikacije). Zmogljivost podatkovnih strežnikov testnega okolja in diskovja je enaka produkcijskim. Kot orodje za vrednotenje zmogljivosti diskovij smo izbrali Diskpsd [9] (za ta namen priporočeno orodje podjetja Microsoft). Nastavljivi parametri izvajanja so bili:

- delež bralnih operacij (ostalo so pisalne);
- število niti, ki vzporedno izvaja operacije;
- trajanje izvajanja testa;
- velikost V/I zahtev (podatkovnih paketov v bajtih);

- delež operacij, ki berejo ali zapisujejo podatke na diskovje na naključnih lokacijah (ostalo so operacije ki berejo ali zapisujejo podatke na diskovje na zaporednih lokacijah).

Med različnimi možnostmi nastavitev smo se odločili za tako, ki bi najbolje odražala način uporabe diskovja kot ga uporablja Microsoft SQL Server 2008 [14].

V tabeli 2.1 in tabeli 2.2 je razvidna primerjava obeh diskovij glede na (obče uporabljane) performančne kazalce testa, ki se izvaja 30 sekund. Več o izbranih parametrih in kazalcih testa je opisano v [14].

| **Kazalec** | **Obstoječe diskovje** | **Diskovje SDD** |
|---|---|---|
| Bytes | 100.597.760 | 10.087.841.792 |
| I/Os | 12.280 | 1.231.426 |
| MB/s | 3,20 | 320,68 |
| I/O per s | 409,28 | 41.046,51 |
| AvgLat | 75,87 | 0,54 |
| LatStdDev | 56,26 | 0,36 |

Tabela 2.1:Rezultati bralnih operacij

| **Kazalec** | **Obstoječe diskovje** | **SDD** |
|---|---|---|
| Bytes | 34.095.104 | 3.367.288.832 |
| I/Os | 4162 | 411.046 |
| MB/s | 1,08 | 107,04 |
| I/O per s | 138,72 | 13.701,19 |
| AvgLat | 6,94 | 0,71 |
| LatStdDev | 41,31 | 0,36 |

Tabela 2.2: Rezultati pisalnih operacij

Za eno izvedbo testa prikazuje kazalec:

- Bytes število prenesenih bajtov;
- I/Os število vhodno izhodnih operacij;
- MB/s povprečno hitrost prenosa podatkov (v MB/s);
- I/O število vhodno izhodih operacij na sekundo;

- AvgLat povprečen latenčni čas;
- LatStdDev povprečno standardno deviacijo latenčnega času.

Vrednosti merjenih kazalcev so pri diskovju SSD boljše vsaj za faktor 100. Operacije z diskovjem so samo del operacij na podatkovnem strežniku. Na podlagi teh rezultatov sicer ni mogoče sklepati, da bi se v povprečju podatkovne operacije podatkovnega strežnika Aplikacije za toliko pohitrile, vendar si je mogoče obetati nezanemarljivo povečanje hitrosti. Povprečni delež trajanja podatkovnih podoperacij v okviru odjemalčevih zahtev je 35% (pri operacijah z uporabniškega seznama ocenjen kot še večji). Izmenjava podatkov med podatkovnim strežnikom in diskovjem sicer ne predstavlja vsega porabljenega časa za podatkovne operacije, ampak njegov večji delež.

Hitrejše izvajanje podatkovnih operacij bi tudi zmanjšalo verjetnost nastopa dogodkov čezmerno nizke odzivnosti (obseg zmanjšanja je vnaprej težko oceniti).

Treba je omeniti, da sodi izposojeno diskovje SSD po podatkovni kapaciteti, izmerjeni performančni zmogljivosti in ostalih karakteristikah v višji razred. Glede na izkazano razmerje med kazalci obstoječega diskovja in diskovja SSD je vodstvo Organizacije sprejelo odločitev o čimprejšnjem nakupu diskovja SSD (upoštevajoč ostale proračunske obveze).

### 2.4.3 Particioniranje pogledov v podatkovni bazi

Na podatkovnem strežniku SQL particionirani pogledi (angl. *partitioned views*) omogočajo zamenjavo (v smislu števila zapisov) velike tabele (v nadaljevanju T) z več manjšimi $T_i$; i=1..n. Tabele $T_i$ so strukturno (shematsko) enake tabeli T. Unija podatkov tabel $T_i$ je enaka vsebini tabele T. Presek podatkov tabel $T_i$ je prazen (vsak zapis samo v eni). Za razporeditev zapisov iz tabele T v $T_i$ mora obstajati jasno določen kriterij porazdelitve podatkov med tabele $T_i$ (v nadaljevanju kriterij vsebovanja). Ta praviloma sloni na konkretnih vrednostih izbranega stolpca. Na podlagi kriterija vsebovanja mora biti znano v katero od tabel $T_i$ sodi nek zapis.

Po izdelavi tabel $T_i$ se definira particioniran pogled (kot unija tabel $T_i$), ki nudi naslednje prednosti:

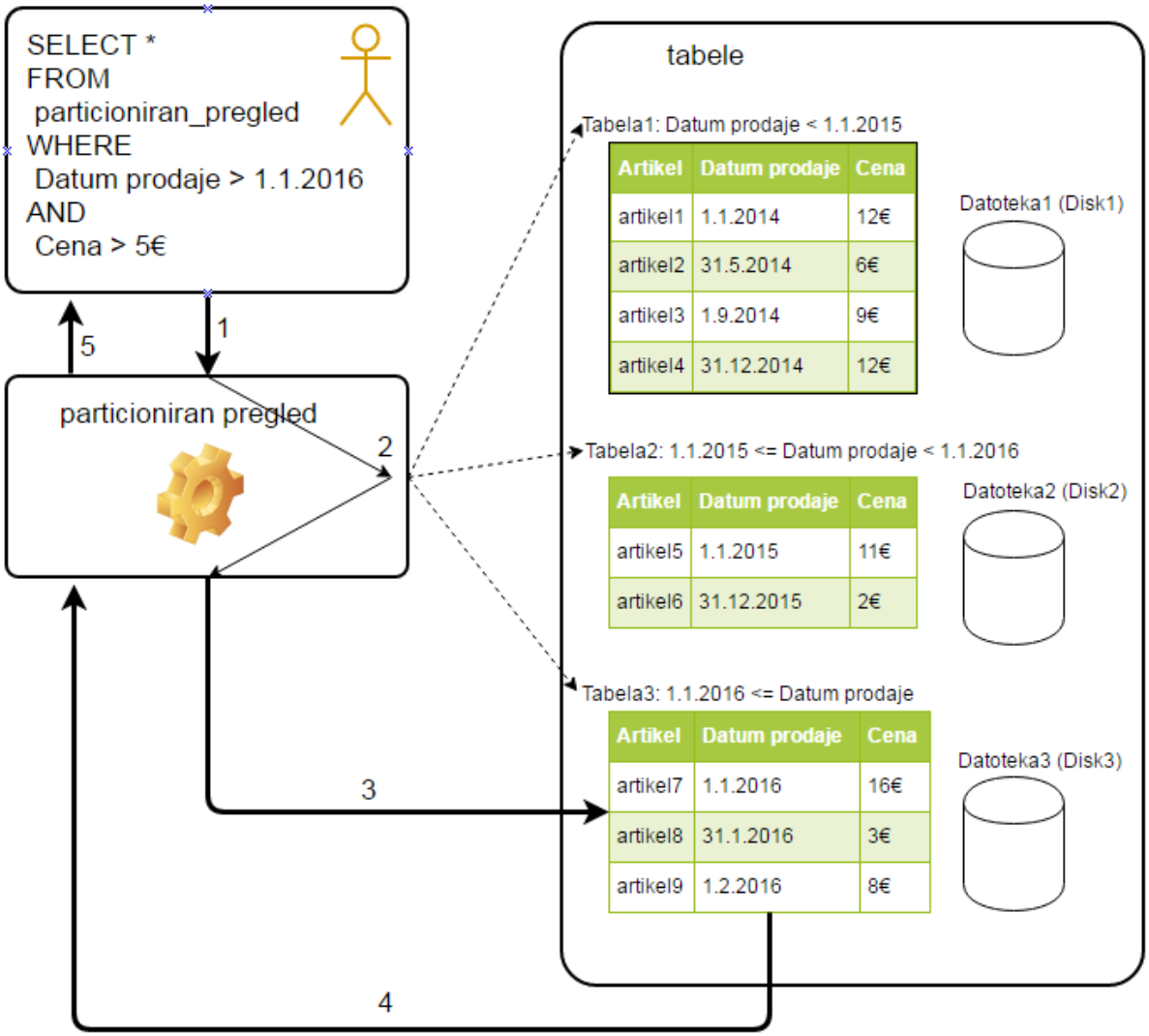
- podatkovne operacije nad manjšimi tabelami so hitrejše kot nad velikimi;
- strežnik SQL zna optimizirati izvedbo podatkovne operacije tako, da se izvede samo nad tabelami, v katerih so ciljani podatki;

- uporabniku  in razvijalcu podrobnosti oz. obstoja večjega števila manjših tabel namesto ene velike ni treba poznati.

Na sliki 2.14 je s konkretnim primerom izvajanja poizvedbe prikazana shema delovanja particioniranega pregleda.

Na podlagi zahteve po seznamu prodanih artiklov v letu 2016, ki so bili prodani za vsaj 5€ (1), strežnik SQL na podlagi pregledanih kriterijev vsebovanja (2) ugotovi, da so iskani zapisi lahko le v tabeli $T_3$. Operacija iskanja zapisov (3) se zato izvede samo na tej tabeli. Izvedba operacije je tako lahko bistveno hitrejša kot v primeru obstoja samo ene velike tabele.

Z uvedbo particioniranega pogleda ni nujno vsaka podatkovna operacija hitrejša. Če v pogoju (WHERE) podatkovne operacije ni uporabljen stolpec kriterija porazdelitve (v primeru na sliki 2.14 je to »datum prodaje«), strežnik SQL ne more optimizirati izvedbe podatkovne operacije (z omejitvijo na eno ali del tabel). V tem primeru se podatkovna operacija izvede nad vsako od tabel in particioniran pogled poskrbi, da se rezultati združeni vrnejo naročniku zahtevka. Trajanje je v tem primeru odvisno tudi od izkoriščenih možnosti paralelnega izvajanja. Strežnik SQL omogoča paralelno izvajanje podatkovnih operacij, če se te izvajajo nad različnimi tabelami in če so te shranjene v ločenih datotekah (primer na sliki 2.14).




Slika 2.14: Shema delovanja particioniranega pregleda

Pri določanju kriterija vsebovanja je treba upoštevati:

- prisotnost stolpcev v pogojih operacij (WHERE);
- porazdelitev tabel na datoteke (npr. vse v eni, vsaka svojo, selektivno grupiranje …);
- porazdelitev datotek na diskovne enote (npr. vse v eni, vsaka svojo, selektivno grupiranje …);
- porazdelitev indeksov (npr. enaki indeksi na vseh tabelah, različni indeksi glede na frekvenco in naravo dostopov do posamične tabele …);

- stroške vzdrževanja (npr. enkratno definiranje tabel in kriterijev vsebovanja, občasni popravki kriterijev vsebovanja še posebej če so datumski, morebiten prenos zapisov med tabelami v primeru občasnega popravljanja kriterijev vsebovanja …);

- stopnjo potrebne pohitritve in naravo dostopov do tabele, ki je predmet particioniranja;

- izkušnje in priporočila oziroma strokovne usmeritve.

V Aplikaciji je nekaj velikih tabel in nekatere med njimi se zdijo primerne za particioniran pogled. Zaradi previdnosti smo se odločili, da se particioniran pogled uvede najprej na eni. Naknadne izkušnje bodo izhodišče za določitev ostalih tabel. Izbrana tabela je ena najbolj kritičnih transakcijskih tabel v smislu števila in hitrosti dnevnih izvedb podatkovnih operacij v tej tabeli. V tabeli je 70 milijonov zapisov, 155 stolpcev in zaseda 90 GB prostora na disku. Indeksi za to tabelo zasedajo dodatnih 83 GB. Vsak zapis je opremljen z datumom poslovnega dogodka. Uporabnikom morajo biti na razpolago vsi zapisi. Največjo frekvenco dostopov imajo zapisi z datumom tekočega meseca. Za razred nižjo frekvenco imajo dostopi do zapisov s poslovnim datumom tekočega leta (brez tekočega meseca). Še za razred nižjo frekvenco imajo dostopi do zapisov, ki imajo poslovni datum, starejši od tekočega leta. V vsakem izmed treh navedenih časovnih okvirov se najpogostejši načini dostopa (podatki v iskalnih kriterijih, tip podatkovne operacije, iskani podatki …) med seboj razlikujejo. Upoštevajoč navedeno smo se odločili za izvedbo particioniranega pogleda tabele na način, da se tabela T razbije v tri manjše $T_1$, $T_2$ in $T_3$. Tabela $T_1$ vsebuje zapise tekočega meseca, $T_2$ zapise tekočega leta brez tekočega meseca in $T_3$ zapise starejše od tekočega leta. Ker smo po novem dobili tri tabele, pri čemer ima vsaka svoj prevladujoč načini dostopa, smo lahko na vsaki definirali indekse bolj optimalno. Seveda je treba vsak konec meseca zapise iz $T_1$ premakniti v $T_2$ in vsak konec leta zapise iz $T_2$ premakniti v $T_3$ in ob premikih popraviti kriterij vsebovanja (saj vsebuje navedbo konkretnega datuma). Smiselno je ta postopek avtomatizirati.

Uvedba particioniranega pogleda poleg hitrostne izboljšave prinaša tudi manjšo verjetnost pojavnosti dogodkov čezmerno nizke odzivnosti, ki so bili prej pogosto vezani na to tabelo. Procesi dnevnega vnosa podatkov (redne operacije) in izdelava poročil za trenutni mesec (redne operacije) se izvajajo nad zapisi v $T_1$, pri čemer ni medsebojnega vpliva s procesi mesečnega obračuna (izredne operacije) in izdelave večmesečnih poročil (izredne operacije), ki se izvajajo nad zapisi v $T_2$. Občasni vpogledi in še nekaj drugih, redkeje izvajanih procesov (vsi izredne operacije), se izvajajo nad zapisi v $T_3$. Operacije nad tabelo $T_1$ praviloma ne vplivajo na hkrati izvajajoče operacije nad tabelo $T_2$ ali $T_3$ (in obratno).

## 2.5 Pohitritev operacij z uporabniškega seznama

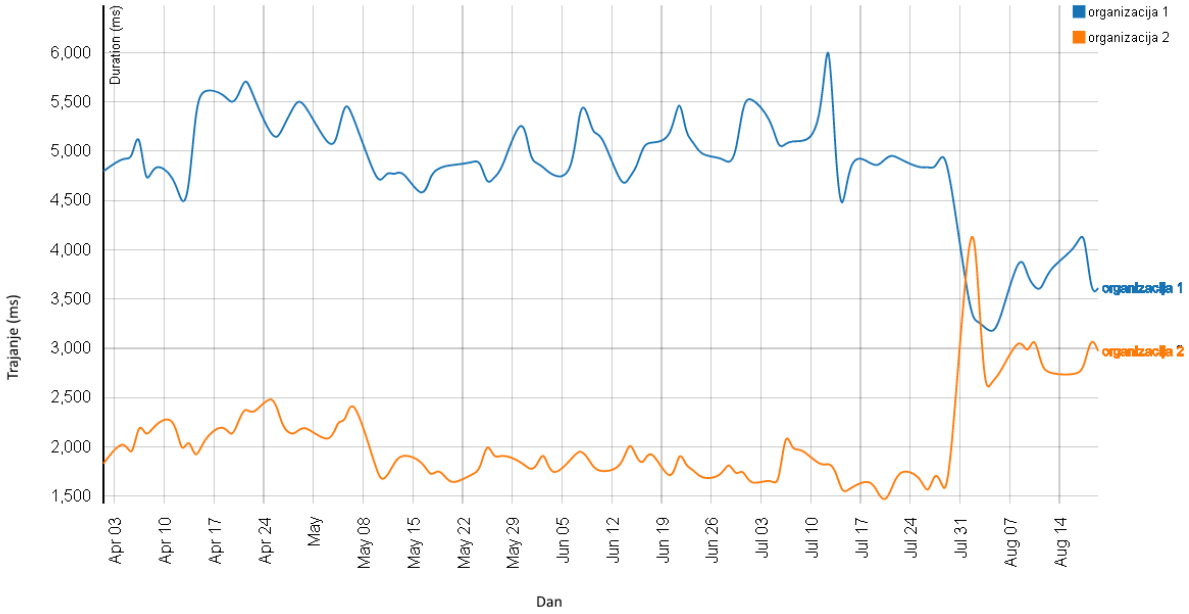
Na podlagi pogovorov z izbranimi uporabniki smo ugotovili, da jih izvedbe operacij z več čakanja (razliko med dejanskim trajanjem operacije in zadostno odzivnostjo) bolj motijo kot tiste z manj čakanja, čeprav z bistveno večjo frekvenco izvajanja. Zato pri določanju prioritetnega vrstnega reda performančne optimizacije operacij z uporabniškega seznama nismo upoštevali pogostosti izvajanja, ampak samo razliko med povprečno dnevno odzivnostjo (iz meritev operacij) in zadostno odzivnostjo (uporabniški seznam).

Pri operacijah z uporabniškega seznama smo sledili priporočenemu postopku [3]:

- okvirna postavitev ciljev (zadostna odzivnost z uporabniškega seznama);
- merjenje odzivnosti sistema pred spremembami (produkcijske meritve operacij);
- identifikacija delov, ki so kritični za izboljšavo (analiza razvijalca);
- merjenje odzivnosti sistema po spremembi (produkcijske meritve operacij);
- sprejem ali zavrnitev sprememb (presoja razvijalca);
- nadaljnje sprotno spremljanje odzivnosti (že opisan proces spremljanja).

Izjema v izvajanju postopka je bila ta, da smo najprej izvedli prvi dve alineji za vse operacije z namenom pridobitve prioritetnega vrstnega reda. Avgusta 2016 še vedno izvajamo ostale štiri alineje po pridobljenem prioritetnem vrstnem redu.

Na sliki 2.15 je primer izboljšave odzivnosti izbrane operacije s seznama. Na grafu je za dve organizaciji prikazano povprečno dnevno trajanje operacije. Vrednosti iz ostalih organizacij zaradi boljše nazornosti niso prikazane na grafu.




Slika 2.15:Prikaz izboljšave odzivnosti izbrane operacije

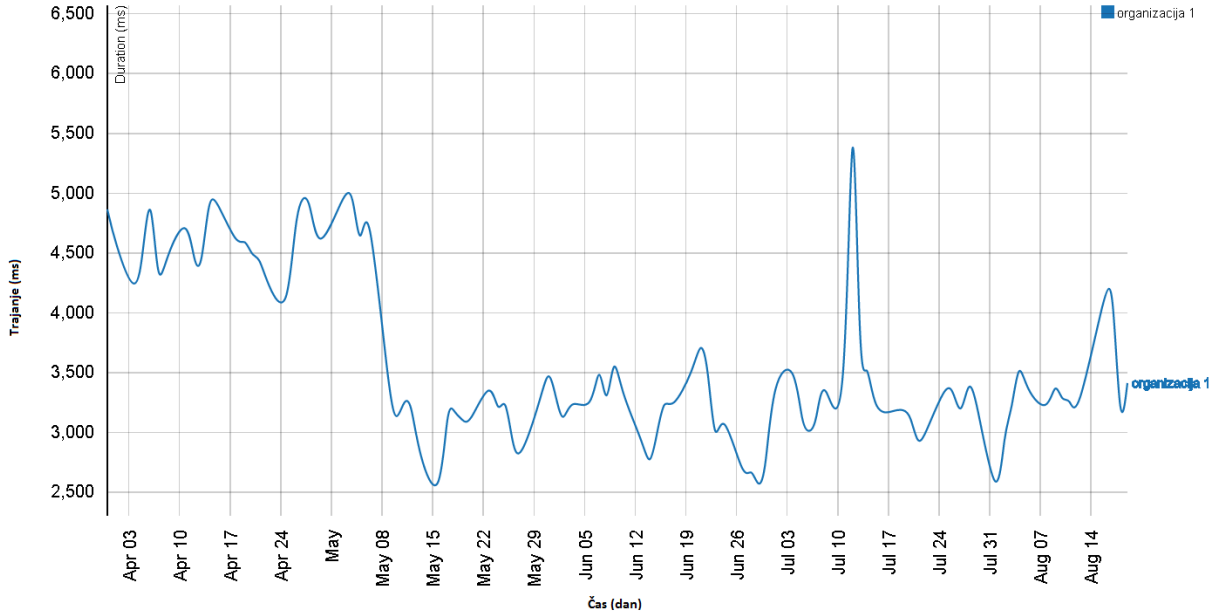
S slike 2.15 je razvidno, da je odzivnost operacije v Organizaciji (organizacija 1) v primerjavi z drugo organizacijo (organizacija 2) občutno slabša. Razvidno je tudi, da je konec julija z novo različico prišlo do upada povprečnega dnevnega časa trajanja (oziroma izboljšanja odzivnosti). Hkrati (primer kompleksnosti specifik okolja), je pri drugi organizaciji razvidno, da je z namestitvijo nove različice prišlo do občutnega poslabšanja odzivnosti, to je trajalo skoraj dva dni. Vzrok poslabšanja ni povezan z optimizacijo, ampak z razširitvijo funkcionalnosti operacije. Za optimalno razširitev je namreč nujen ustrezen indeks na določeni tabeli. Indeks je bil naknadno izdelan in odzivnost se je popravila, vendar ne na nivo predhodne različice. Zadeva čaka na podrobnejšo analizo.

Tovrstno spremljanje (podrobneje opisano v poglavju 3) omogoča vpogled v spremembe v odzivnosti, ki jih prinašajo nove različice, čeprav te niso edini možen vzrok takih sprememb.

Na sliki 2.16 je orisan še en primer izboljšave odzivnosti operacije. Z grafa je razvidno, da je v prvi tretjini maja prišlo do izboljšanja odzivnosti. Vzrok izboljšanja je optimizacija skripte SQL na podatkovni bazi.

Izbira ustrezne metode za pohitritev izbrane podoperacije je odvisna od fizične lokacije izvajanja. Če je predmet optimizacije izvedba podoperacije na podatkovnem strežniku potem se poskuša optimizirati izvedbo skripte SQL (npr. uporaba dinamičnega SQL-a, začasnih vmesnih tabel …) in po potrebi dodati ustrezen indeks na problematično tabelo. Če je predmet optimizacije podoperacija na odjemalcu ali aplikacijskem strežniku, je prevladujoča oblika

izbira hitrejšega algoritma in uporaba predpomnjenja (angl. *caching*) ter vnaprejšnja priprava konteksta izvedbe operacije (npr. vnaprejšnja pridobitev dela podatkov). Če se želi pohitriti prenos velikega sporočila, se uporabi kompresijo (dokler generičen postopek kompresije po postopku 2.4.2.1 ne bo uspešno implementiran je mogoče kompresijo narediti na višjem nivoju, specifično za izbrano sporočilo). Pri manjšem številu operacij smo ugotovili, da uporabnik za takojšnje nadaljevanje svojega dela ne potrebuje rezultatov operacije. Zato smo pri teh operacijah način izvedbe spremenili tako, da se izvedba zažene na ločeni niti »v ozadju« (angl. *background thread*), pri čemer se uporabniški vmesnik ne zaklene. Uporabnik v tem primeru lahko takoj nadaljuje z delom (ni mu treba čakati na konec izvedbe). Take operacije smo umaknili iz postopka spremljanja odzivnosti.




Slika 2.16: Primer izboljšane odzivnosti še ene operacije

Proces izboljševanja odzivnosti operacij s seznama je v začetni fazi. Do sredine avgusta 2016 so bile izboljšane 3 operacije. Iz navedenega je razvidno, kako poteka izboljševanje odzivnosti operacij z uporabniškega seznama. Proces izboljševanja je postopen, saj ga je treba usklajevati z drugimi nalogami.

# Poglavje 3 Implementacija sprotnega spremljanja odzivnosti

Zaradi rednih namestitev novih različic je vedno mogoče (in nezaželeno), da pride do poslabšanja odzivnosti ene ali več operacij zaradi neoptimalnih sprememb izvorne kode. Če primerjamo zaporedni različici, so to lahko tudi manjša (težko opazna) poslabšanja. V daljšem obdobju (večje število novih različic) je pri določeni operaciji znižan nivo odzivnosti lahko že moteč. Vzroki poslabšanja so lahko tudi dogodki sprememb v okolju Aplikacije, kot npr. menjava diskovja, povečana obremenitev skupnih virov zaradi drugih porabnikov, večje število uporabnikov … Med pogostimi vzroki, ki niso vezani na novo namestitev in tudi ne na okolje Aplikacije, je doseganje kritične meje količine podatkov v neki tabeli. Kritičnost v kontekstu, da obstoječi indeks(-i) glede na velikost tabele ne nudi(-jo) več optimalne podlage za hitro izvedbo operacij. V takih primerih je treba dopolniti definicijo indeksa(-ov) na način, ki je bolj optimalen za trenutno količino podatkov.

V primeru izvajanja analize poslabšanja odzivnosti neke operacije, ko je od pojava vzrokov preteklo že daljše obdobje (mesec ali več), je težko uspešno identificirati vzročni dogodek. Zaradi časovne oddaljenosti se možne vzročne dogodke ne uspe identificirati saj se nanje lahko preprosto pozabi. Tudi pri identificiranih dogodkih je brez ustreznih meritev odzivnosti pred pojavom dogodka, morda težko ali nemogoče potrditi vzročno povezavo, zato smo se odločili vpeljati trajno spremljanje nekaterih reprezentativnih operacij (npr. z uporabniškega seznama). Merjenje se izvaja ves čas. Spremljanje meritev se v primeru namestitve nove različice izvaja dnevno (v trajanju nekaj dni), sicer tedensko. Za spremljanje so odgovorni tudi v Organizaciji (in ostalih organizacijah).

Za ustrezno podporo procesu spremljanja odzivnosti je bilo treba omogočiti:

- merjenje odzivnosti vseh izvedb vnaprej izbranih operacij;
- enostavno dodajanje merjenja odzivnosti novih operacij;
- merjenje odzivnosti tudi v ostalih organizacijah;
- grafičen prikaz meritev;
- prikaz meritev vnaprej določenih operacij na istem grafu;
- prikaz meritev vnaprej določenih operacij vseh organizacij na istem grafu;

- prikaz meritev vnaprej določenih operacij za obdobje zadnjih 1000 minut;
- prikaz meritev vnaprej določenih operacij za obdobje zadnjih 1000 ur;
- prikaz meritev vnaprej določenih operacij za obdobje zadnjih 1000 dni;
- samodejno obveščanje odgovornih če odzivnost vnaprej določenih operacij preseže vnaprej določene časovne meje.

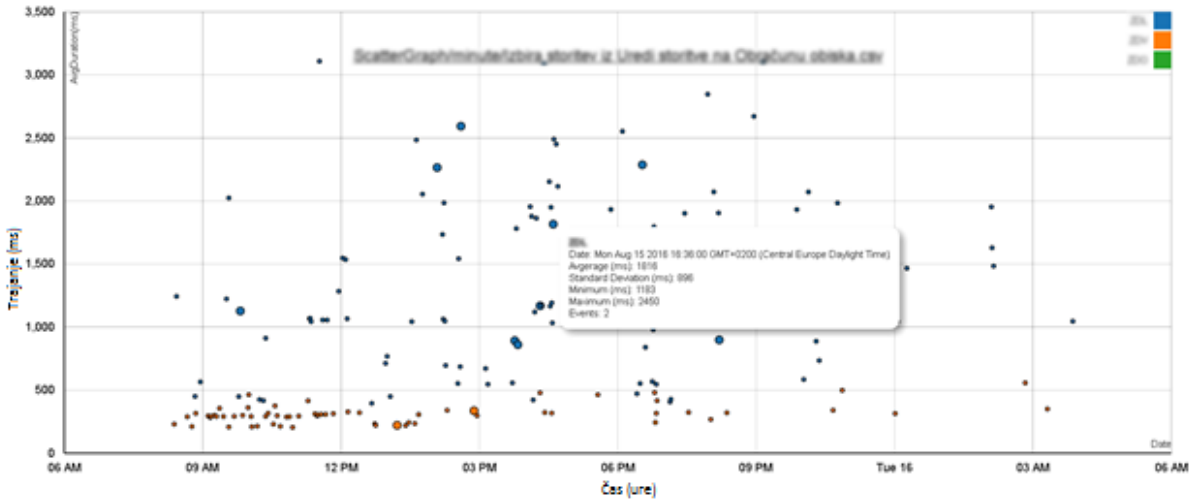
Ocenjena odprtokodna in komercialna orodja za izvajanje in spremljaje meritev (*Cacti[13], Nagios[11], Icinga[10], Zabbix[16] ...*) so predraga, prekompleksna ali pomanjkljiva, zato smo se odločili za lastno implementacijo na osnovi programskega skriptnega jezika *PowerShell [12]* in vizualizacijske knjižnice *d3js [8]*.

Za izdelavo prikaza se izvaja:

- prevzem podatkov:
    - za obdobje zadnjih 1000 minut na vsakih 5 minut;
    - za obdobje zadnjih 1000 ur na vsako uro;
    - za obdobje zadnjih 1000 dni enkrat dnevno.
- preverjanje podatkov:
    - preseganje določenih časovnih mej;
    - obveščanje odgovornih o preseganju.
- organizacija podatkov
    - glede na časovni interval;
    - glede na tip grafa;
    - glede na organizacijo.
- namestitev podatkov
    - na spletne strežnike  vsake organizacije.

Vsaka organizacija ima na lastnem spletnem strežniku objavljeno spletno stran, na kateri je mogoč vpogled v grafe. Na sliki 3.1 je razviden graf raztrosa podatkov operacije za več organizacij (za medsebojno primerjavo). Različni grafi imajo različne intervale (1000 minut, ur, dni). Na sliki 3.1 je merjeni interval minuta. Z grafa je poleg časa (s časovne osi) mogoče razbrati še:

- povprečno trajanje znotraj intervala (y os);
- povprečno trajanje operacij;
- standardno deviacijo trajanja operacij;
- minimalen čas trajanja;
- maksimalen čas trajanja;
- število izvedb operacije.




Slika 3.1: Posnetek spletne strani za izbran graf raztrosa

Primeri linijskih grafov so razvidni s slik 2.15 in 2.16. Ti so primernejši za spremljanje poteka odzivnosti za daljše obdobje.

# Poglavje 4 Reševanje dogodkov čezmerno nizke odzivnosti

## 4.1 Odpravljanje vzrokov

V poglavju 1.4.3 so identificirani glavni vzroki dogodkov čezmerno nizke odzivnosti in smernice reševanja. Ugotovljeno je, da bolj optimalna izvedba nekaterih nerednih operacij ni možna ali smiselna. Pri nekaterih nerednih operacijah, kjer je bolj optimalen način izvedbe možen in smiseln, ta ne bo vedno zagotavljal neproblematične obremenitve virov podatkovnega strežnika. Upoštevajoč še dejstvo, da je težko identificirati vse možne težavne kombinacije nerednih operacij, je preprečitev vseh možnih težavnih kombinacij zaradi ozkih grl (v virih podatkovnega strežnika) nedosegljiva. Nakup diskovja SSD (poglavje 2.4.2) bo pohitril izvajanje nerednih (in rednih) operacij in zmanjšal medsebojni vpliv, vendar ne more zagotavljati, da se taki dogodki ne bodo pojavili.

Izvrševanje neredne operacije se lahko prekine, če moteče vpliva na redno delo uporabnikov. Pojave nizke odzivnosti se zato rešuje z ustavitvijo neredne operacije. Težava je velikokrat v hitrosti obveščanja odgovornih. Dogaja se, da uporabniki pri pojavu nizke odzivnosti ne sporočijo tega takoj prvemu nivoju podpore. Vzroki so lahko različni (npr. »gotovo bo/je že sporočil nekdo drug podpori«, »saj bo najbrž kmalu spet normalno delovalo« ...). Ko je o tem obveščen prvi nivo podpore (služba Organizacije), ta najprej preveri morebitne vzroke v lastni infrastrukturi (mrežne povezave, strežniki ...). Če je ugotovljeno, da vzroki niso infrastrukturne narave, se težavo prijavi na drug nivo podpore (izvaja ga razvijalec/vzdrževalec Aplikacije).

## 4.2 Preprečevanje dogodkov čezmerno nizke odzivnosti

Velika večina nerednih operacij ni nujne narave in se lahko izvede z zamikom (v nočnem času), ko ni vpliva na redno delo velike večine uporabnikov zaradi njihove odsotnosti. Zato smo pri zagonu identificiranih potencialno spornih (dlje trajajočih, zahtevnih do skupnih virov) nerednih operacijah dodali ustrezno opozorilo o morebitnem vplivu takojšnjega izvajanja na ostale uporabnike in možnost naročila izvedbe z zamikom (npr. na nočni čas). Uporabnik se za tak zamaknjen način lahko odloči ali ne.

Pri večjem številu spornih kombinacij se je kot vzrok pokazala hkratna (in zaradi neredne operacije tudi zahtevnejša) uporaba ene izmed največjih tabel v podatkovni bazi. Hkraten dostop do tega skupnega vira je bil pogosto ozko grlo. Na podlagi narave težave smo se odločili

za pristop particioniranja pogledov (podrobneje opisan v poglavju 2.4.3), pri katerem se v osnovi skupna težavna tabela razbije na več manjših. Pri tem se zanašamo na oceno, da bodo operacije pogosto dostopale samo do ene manjših tabel (pomemben je kriterij razbitja) in da bodo pri hkratni izvedbi dostopale vsaka do svoje tabele. Posledično pomeni to hitrejšo v izvedbo in manjši medsebojni vpliv sočasnih operacij.

## 4.3 Zgodnje zaznavanje dogodkov čezmerno nizke odzivnosti

V večini takih primerov predstavlja ozko grlo podatkovni strežnik. Običajno je takrat večina strežnikove zmogljivosti uporabljena za izvajanje dlje trajajoče zahtevne neredne operacije. Med njenim izvajanjem se ostale podoperacije upočasnijo ali sploh ne izvajajo (angl. *blocking session*). Uporabljana izdaja podatkovnega strežnika (*Microsoft SQL Server Standard Edition)* ne omogoča zahtevnim operacijam nižanja prioritete izvajanja. To omogoča dražja izdaja *Enterprise Edition*, katere nakup zaradi omenjenih težav ni upravičljiv.

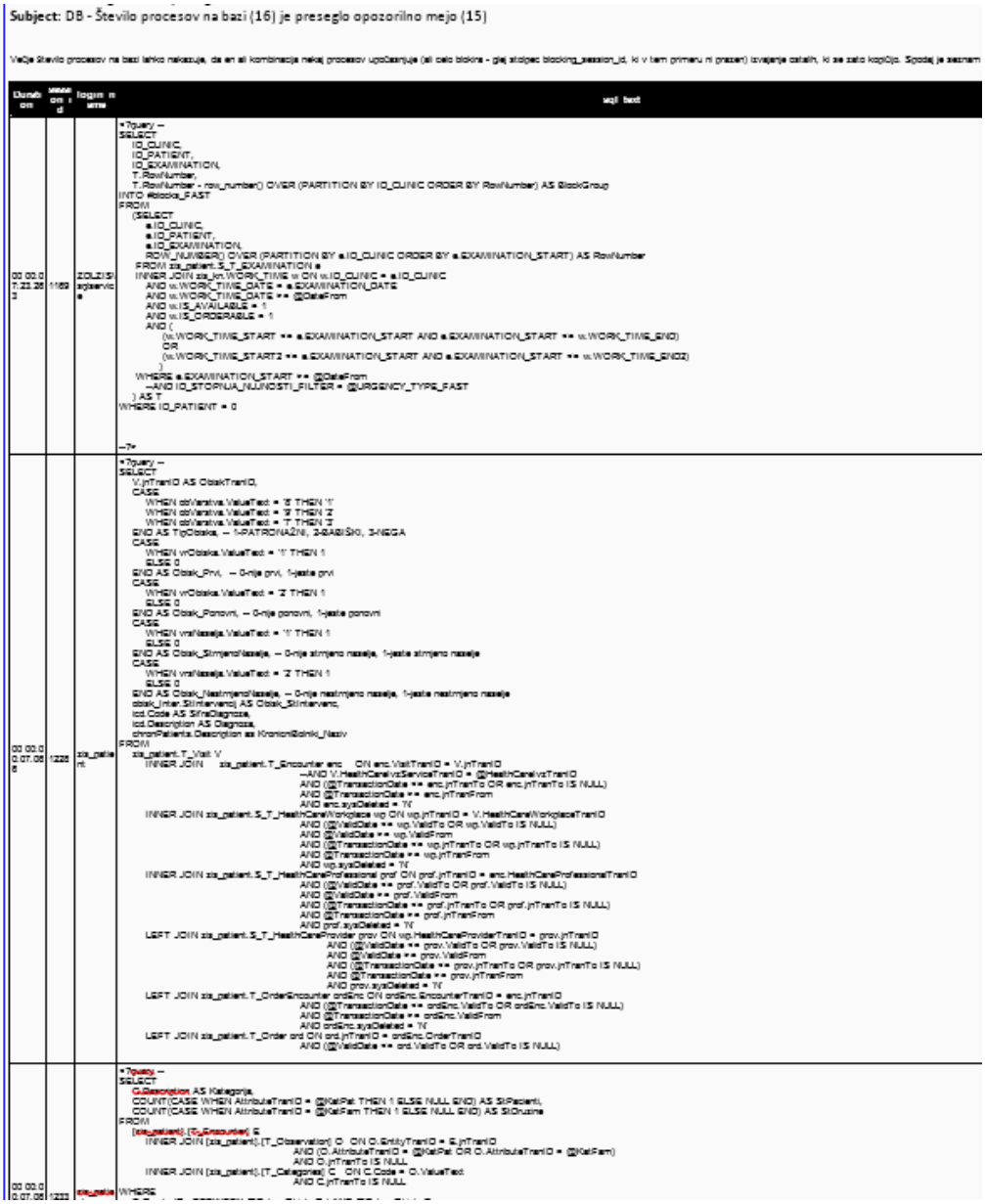
Zaradi manjšega obsega razpoložljivih zmogljivosti se izvajanje drugih podoperacij podaljša tudi na nekaj minut (ali več). Ker še vedno prihajajo nove zahteve za izvedbo podoperacij, se število izvajanih povečuje (čeprav počasi). Glede na dolgoletne izkušnje ocenjujemo, da je neobičajno, če se število sočasno izvajanih podoperacij na podatkovnem strežniku povzpne čez 15. Zato je dogodek preseganja te meje zelo verjeten indikator pojava nastajajočega ozkega grla.

Izvedbo samodejne, (na dve minuti) ponavljajoče kontrole števila trenutno izvajanih podoperacij na podatkovnem strežniku smo imeli za ustrezno reševanje težave. V primeru preseganja vnaprej določene meje sledi obveščanje odgovornih oseb po elektronski pošti. V obvestilu se nahaja seznam izvajanih podoperacij in za vsako še dodaten nabor spremljajočih informacij:

- trajanje (izvajanja podoperacije do tega trenutka);
- enolični identifikator podoperacije (uporabljen pri morebitnem naknadnem ročnem prekinjanju podoperacije);
- izvajajoči stavek SQL (omogoča hitro oceno ali gre za redno ali neredno podoperacijo);
- naziv uporabnika;
- število ostalih podoperacij, ki čakajo na izvedbo te;

- do tega trenutka količina uporabljenih ciklov CPE;
- količina pisanj v strežnikovo začasno tabelo *TempDB* (pomemben skupen vir);
- količina uporabljenih strani začasne tabele;
- enolični identifikator podoperacije, ki povzroča čakanje te;
- do tega trenutka število opravljenih branj (diskovja ali spomina);
- do tega trenutka število opravljenih pisanj;
- do tega trenutka število opravljenih branj z diskovja;
- količina trenutno uporabljenega pomnilnika;
- status (izvajajoča, čakajoča ...);
- število transakcij, ki jih je podoperacija odprla;
- naziv računalnika, ki je podal zahtevo izvedbe podoperacije (običajno aplikacijski strežnik)
- ...

Na sliki 4.1 je primer takega obvestila. Nekoliko manj zanesljiv indikator pojava nastajajočega ozkega grla je izvajanje neke podoperacije na podatkovnem strežniku dlje kot 5 minut. Dodali smo tudi tako redno ponavljajočo kontrolo. Takoj po začetku izvajanja kontrole smo identificirali nekaj primerov podoperacij za katere je pričakovano izvajanje v daljšem trajanju kot 5 minut. Glede na vsebino stavka SQL smo za vsak izstopajoči primer podoperacije določili višjo mejo. Za določene primere podoperacij je mogoče nastaviti mejo tudi nižje. Pri tem je treba paziti, da se tako prilagajanje izvaja sproti. Če prihaja do pogostega pošiljanja lažnih opozoril, postanejo naslovniki nanje manj občutljivi.




Slika 4.1: Obvestilo odgovornim o prekoračitvi nastavljene meje števila hkrati izvajajočih podoperacij na podatkovnem strežniku

# Poglavje 5 Zaključek

V diplomski nalogi smo spoznali in opisali problematiko postopnega upada odzivnosti, dogodkov čezmerno nizke odzivnosti in neobstoječega nadzora odzivnosti konkretne trinivojske aplikacije. Natančneje smo opisali glavne razloge za postopen upad kot sta naraščajoča količina podatkov in kompleksnost izvorne kode. Opredelili smo odzivnost in se osredotočili na uporabniške izkušnje in prioritete. Opisali smo sistem merjenja odzivnosti na treh nivojih (uporabniških operacij, odjemalčevih zahtevkov na aplikacijski strežnik in zahtevkov aplikacijskega strežnika na podatkovni strežnik) in ga implementirali. Analizirali smo rezultate, predlagali in delno uspešno implementirali predloge za boljšo pretočnost po omrežju.

Zastavili in začeli smo izvajati proces postopnega izboljševanja odzivnosti posameznih operacij (in na splošno) na podlagi uporabniških izkušenj in prioritet. Zastavili in začeli smo tudi dolgotrajni proces spremljanja odzivnosti, ki bo zagotavljal učinkovitejše in pravočasno naslavljanje težav vezanih na sicer težko zaznavno postopno slabšanje odzivnosti. Tako smo uspešno zaustavili postopen upad odzivnosti saj takoj po namestitvi nove različice naslovimo morebitno poslabšanje odzivnosti.

Kljub temu je proces izboljševanja odzivnosti operacij z uporabniškega seznama šele v začetni fazi saj je neposredne optimizacije bilo deležno le nekaj primerov. Načrtujemo nadaljevati z delom, ki vključuje vendar ni omejeno na:

- neposredno optimizacijo ostalih operacij z uporabniškega seznama;
- nadaljnje raziskovanje možnosti izvedbe kompresije pred postopkom šifriranja;
- analizo možnosti opustitve obstoječega šifriranja kot ločenega koraka in izvajanja z geslom zaščitene kompresije;
- dodajanje obvestila uporabnikom o morebitnem poteku komunikacije z zunanjim sistemom (sicer uporabnik morebitno počasnost postopka pripiše slabi odzivnosti Aplikacije);
- izvedbo izbranih operacij na ločeni niti v ozadju zato lahko uporabnik takoj nadaljuje z delom;
- optimizacijo indeksov;

- optimizacijo skript SQL;
- uvedbo particioniranja še na druge primerne tabele;
- uvedbo arhiviranja (prenosa) neaktualnih zapisov v arhivske tabele.

Nazadnje smo identificirali vzroke dogodkov čezmerno nizke odzivnosti, v obsegu dosegljivega zmanjšali verjetnost tega pojava in vzpostavili sistem nadzora ter zgodnjega javljanja pojava takih dogodkov. Tako smo v enem mesecu smo dvakrat preprečili razvoj dogodka čezmerno nizke odzivnosti s pravočasno ustavitvijo neredne operacije. To smo storili takoj po prejetju obvestila o preseženem številu operacij na podatkovnem strežniku.

V okviru do zdaj izvedene izboljšave odzivnosti in naslavljanja te problematike smo se naučili, da je treba pri aplikacijah, katerih življenjski cikel narekuje intenziven razvoj novih in razširitve obstoječih funkcionalnosti, posebno pozornost posvetiti rednemu spremljanju in vzdrževanju ustreznega nivoja odzivnosti. Sicer se bo ta zelo verjetno postopno zniževala, kar bo povzročilo nezadovoljstvo uporabnikov. To bo zahtevalo nepredvidene, obsežne in zahtevne posege v aplikacijo. Stroškovno in s stališča kvalitetnih poslovnih odnosov je to zelo nezaželeno.

# Literatura


[1] J. Duffy, *Application responsiveness-Using concurrency can enhance user experience,* Dr Dobbs Journal, okt. 2006, str 59-62.

[2] E. Fernando, D. Touriano in Rico, "Impact of Service-Oriented Architecture adoption in information system," *2015 2nd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, Semarang, 2015, str. 52-55.

[3] R. Mariani, B. Bohling, C. U. Smith, in S. Barber, *Improving .NET Application Performance and Scalability*", Microsoft Corporation, 2004.

[4] T. J. McCabe, "A Complexity Measure," v *IEEE Transactions on Software Engineering*, zv. SE-2, št. 4, str. 308-320, Dec. 1976.

[5] J.D. Meier, D. Hill, A. Homer, J. Taylor, P. Bansode, L. Wall, R. Boucher Jr., A. Bogawat, *Microsoft Application Architecture Guide, 2nd Edition*, Microsoft Corporation, 2009

[6] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, 2003, str. 3-12.6

[7] Chapter 1: Service Oriented Architecture (SOA). MSDN Library [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/bb833022.aspx

[8] D3 Data-Drive Documents [Online]. https://d3js.org/

[9] Diskspd Utility: A Robust Storage Testing Tool [Online]. Dosegljivo: https://gallery.technet.microsoft.com/DiskSpd-a-robust-storage-6cd2f223. [Dostopano 28.7. 2016]

[10] Incinga documentation [Online]. Dosegljivo: http://docs.icinga.org/icinga2/latest/doc/module/icinga2/toc. [Dostopano 28. 7. 2016]

[11] Nagios Log Server Documentation [Online]. Dosegljivo: https://library.nagios.com/library/products/nagios-log-server/documentation. [Dostopano 28. 7. 2016]

[12] Powershell. *Msdn* [Online]. https://msdn.microsoft.com/en-us/powershell/mt173057.aspx

[13] The Cacti Manual [Online]. Dosegljivo: http://www.cacti.net/downloads/docs/html/. [Dostopano 28. 7. 2016]

[14] Using Microsoft DiskSpd to Test Your Storage Subsystem [Online]. Dosegljvo: http://sqlperformance.com/2015/08/io-subsystem/diskspd-test-storage. [Dostopano 28. 7. 2016]

[15] Using Partitioned Views. TechNet [Online]. Dosegljivo: https://technet.microsoft.com/en-us/library/ms190019(v=sql.105).aspx. [Dostopano 28. 2. 2016]

[16] Zabbix manual [Online]. Dosegljivo: http://www.zabbix.com/documentation.php. [Dostopano 28. 7. 2016]